PYIMAGESEARCH

DEEP LEARNING (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/DEEP-LEARNING/)

KERAS AND TENSORFLOW (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/KERAS-AND-TENSORFLOW/)

TUTORIALS (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/)

Keras ImageDataGenerator and Data Augmentation

by **Adrian Rosebrock** (https://www.pyimagesearch.com/author/adrian/) on July 8, 2019

Click here to download the source code to this post

(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_header.png)

In today's tutorial, you will learn how to use Keras' ImageDataGenerator class to perform data augmentation. I'll also dispel common confusions surrounding what data augmentation is, why we use data augmentation, and what it does/does not do.

Knowing that I was going to write a tutorial on data augmentation, two weekends ago I decided to have some fun and purposely post a *semi-trick question* on my **Twitter feed (https://twitter.com/PyImageSearch/status/1142765698575413248)**.

The question was simple — **data augmentation does which of the following?**

1. Adds more training data
2. Replaces training data

3 Does both

4 I don't know

Here are the results:

(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_poll.png)

Figure 1: My **@PyImageSearch (https://twitter.com/pyimagesearch)** **twitter poll (https://twitter.com/PyImageSearch/status/1142765698575413248)** on the concept of Data Augmentation.

Only 5% of respondents answered this trick question "correctly" (at least if you're using Keras'

ImageDataGenerator class).

Again, it's a trick question so that's not exactly a fair assessment, but here's the deal:

While the word *"augment"* means to make something "greater" or "increase" something (in this case, data), the Keras ImageDataGenerator class *actually* works by:

1. Accepting a batch of images used for training.
2. Taking this batch and applying a series of random transformations to each image in the batch (including random rotation, resizing, shearing, etc.).
3. **Replacing the original batch with the new, randomly transformed batch.**
4. Training the CNN on this randomly transformed batch (i.e., the original data *itself* is *not* used for training).

That's right — **the Keras ImageDataGenerator class is *not* an "additive" operation.** It's not taking the original data, randomly transforming it, and then returning *both* the original data and transformed data.

Instead, the ImageDataGenerator accepts the original data, randomly transforms it, and returns *only* the new, transformed data.

But remember how I said this was a trick question?

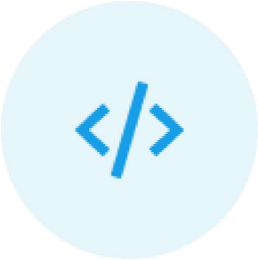
Technically, all the answers are correct — **but the only way you know if a given definition of data augmentation is correct is via the context of its application.**

I'll help you clear up some of the confusion regarding data augmentation (and give you the context you need to successfully apply it).

Inside the rest of today's tutorial you will:

- Learn about three types of data augmentation.
- Dispel any confusion you have surrounding data augmentation.
- Learn how to apply data augmentation with Keras and the `ImageDataGenerator` class.

To learn more about data augmentation, including using Keras' ImageDataGenerator class, *just keep reading!*



Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

Keras ImageDataGenerator and Data Augmentation

2020-06-04 Update: *This blog post is now TensorFlow 2+ compatible!*

We'll start this tutorial with a discussion of data augmentation and why we use it.

I'll then cover the three types of data augmentation you'll see when training deep neural networks:

1. Dataset generation and data expansion via data augmentation (less common)
2. In-place/on-the-fly data augmentation (most common)
3. Combining dataset generation and in-place augmentation

From there I'll teach you how to apply data augmentation to your own datasets (using all three methods) using Keras' `ImageDataGenerator` class.

What is data augmentation?

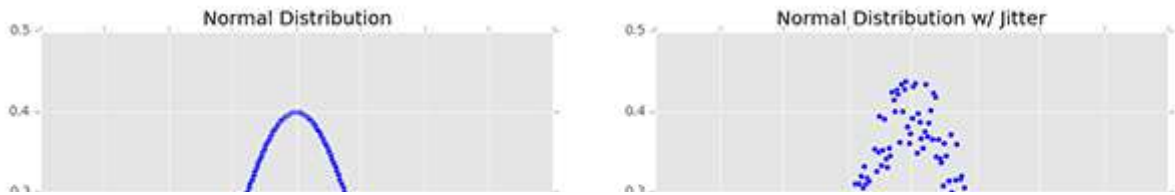
Data augmentation encompasses a wide range of techniques used to generate "new" training samples from the original ones by applying random jitters and perturbations (but at the same time ensuring that the class labels of the data are not changed).

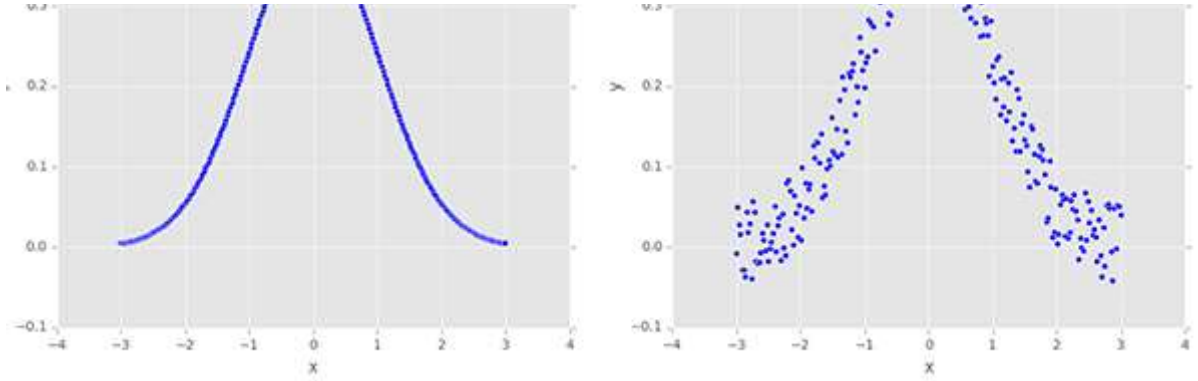
Our goal when applying data augmentation is to increase the generalizability of the model.

Given that our network is constantly seeing new, slightly modified versions of the input data, the network is able to learn more robust features.

At testing time we do not apply data augmentation and simply evaluate our trained network on the unmodified testing data — in most cases, you'll see an increase in testing accuracy, perhaps at the expense of a slight dip in training accuracy.

A simple data augmentation example





(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_jitter.png)

Figure 2: ***Left:*** A sample of 250 data points that follow a normal distribution exactly. ***Right:*** Adding a small amount of random "jitter" to the distribution. This type of data augmentation increases the generalizability of our networks.

Let's consider **Figure 2** (*left*) of a normal distribution with zero mean and unit variance.

Training a machine learning model on this data may result in us modeling the distribution *exactly* — however, in real-world applications, data rarely follows such a nice, neat distribution.

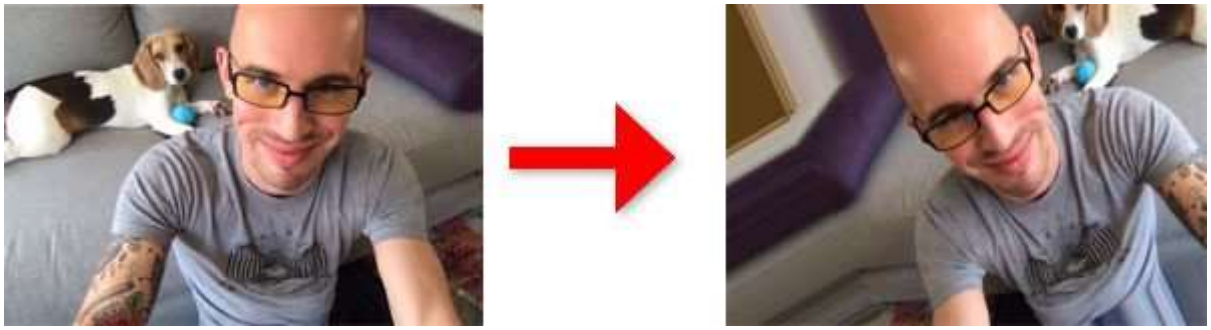
Instead, to increase the generalizability of our classifier, we may first randomly jitter points along the distribution by adding some random values ϵ drawn from a random distribution (*right*).

Our plot still follows an *approximately* normal distribution, but it's not a *perfect* distribution as on the left.

A model trained on this modified, augmented data is more likely to generalize to example data points not included in the training set.

Computer vision and data augmentation

Computer vision and data augmentation



(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_random_adjustments.jpg)

Figure 3: In computer vision, data augmentation performs random manipulations on images. It is typically applied in three scenarios discussed in this blog post.

In the context of computer vision, data augmentation lends itself naturally.

For example, we can obtain augmented data from the original images by applying simple geometric transforms, such as random:

1. Translations
2. Rotations
3. Changes in scale
4. Shearing

5 Horizontal (and in some cases, vertical) flips

Applying a (small) amount of the transformations to an input image will change its appearance slightly, but it *does not* change the class label — thereby making data augmentation a very natural, easy method to apply for computer vision tasks.

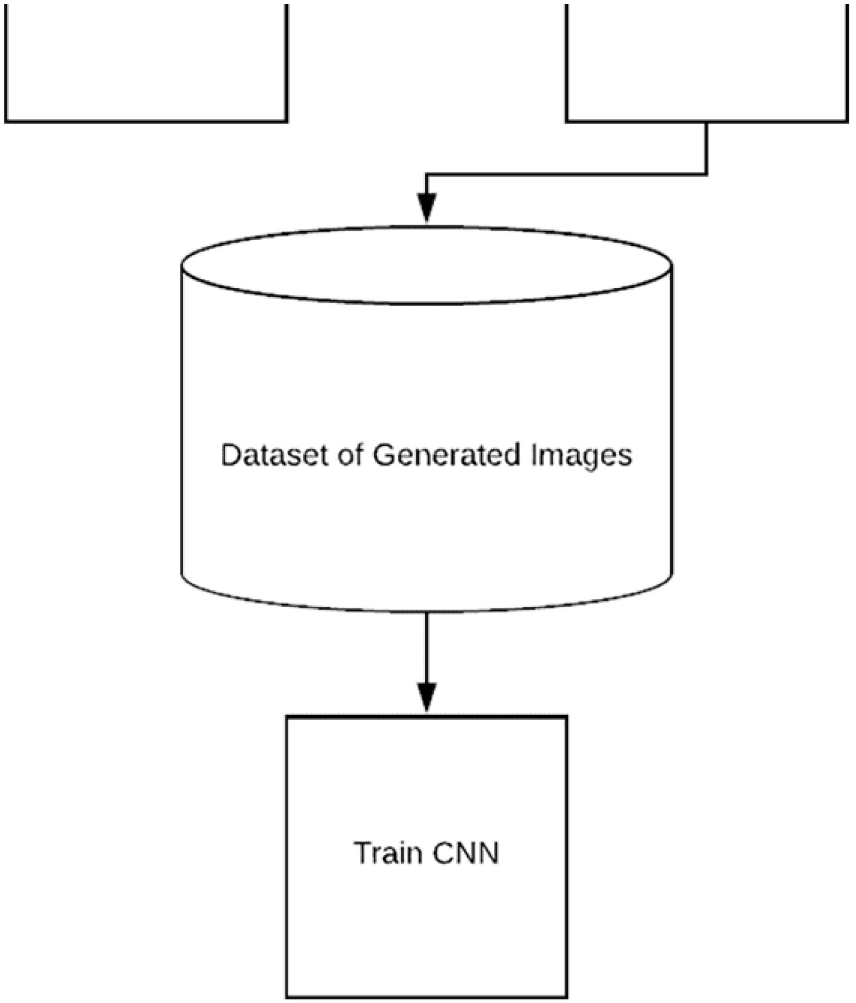
Three types of data augmentation

There are three types of data augmentation you will likely encounter when applying deep learning in the context of computer vision applications.

Exactly which definition of data augmentation is "correct" is *entirely dependent* on the context of your project/set of experiments.

Take the time to read this section *carefully* as I see many deep learning practitioners confuse what data augmentation does and does not do.

Type #1: Dataset generation and expanding an existing dataset (less common)



(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_dataset_generation.png)

Figure 4: Type #1 of data augmentation consists of dataset generation/dataset expansion. This is a less common form of data augmentation.

The first type of data augmentation is what I call **dataset generation** or **dataset expansion**.

As you know machine learning models, and especially neural networks, can require quite a bit of training data — **but what if you don't have very much training data in the first place?**

Let's examine the most trivial case where you only have *one image* and you want to apply data augmentation to create an entire dataset of images, all based on that one image.

To accomplish this task, you would:

1. Load the original input image from disk.
2. Randomly transform the original image via a series of random translations, rotations, etc.
3. Take the transformed image and write it back out to disk.
4. *Repeat steps 2 and 3 a total of N times.*

After performing this process you would have a directory full of randomly transformed "new" images that you could use for training, all based on that single input image.

This is, of course, an incredibly simplified example.

You more than likely have more than a single image — you probably have 10s or 100s of images and now your goal is to turn that smaller set into 1000s of images for training.

In those situations, dataset expansion and dataset generation may be worth exploring.

But there's a problem with this approach — we haven't exactly increased the ability of our model to generalize.

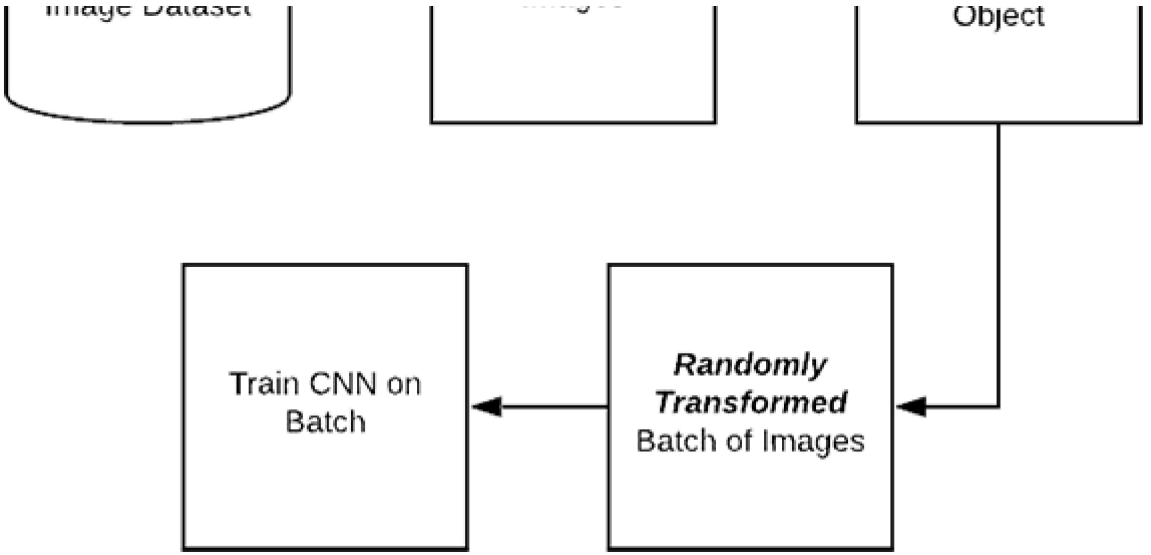
Yes, we have increased our training data by generating additional examples, but all of these examples are based on a *super small* dataset.

Keep in mind that our neural network is only as good as the data it was trained on.

We *cannot* expect to train a NN on a small amount of data and then expect it to generalize to data it was *never* trained on and has never seen before.

If you find yourself seriously considering dataset generation and dataset expansion, you should take a step back and instead invest your time gathering additional data or looking into methods of **behavioral cloning** (and then applying the type of data augmentation covered in the *"Combining dataset generation and in-place augmentation"* section below).

Type #2: In-place/on-the-fly data augmentation (most common)



(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_in_place.png)

Figure 5: Type #2 of data augmentation consists of on-the-fly image batch manipulations. This is the most common form of data augmentation with Keras.

The second type of data augmentation is called **in-place data augmentation** or **on-the-fly data augmentation**. This type of data augmentation is what Keras' `ImageDataGenerator` class implements.

Using this type of data augmentation we want to ensure that our network, when trained, sees new variations of our data *at each and every epoch*.

Figure 5 demonstrates the process of applying in-place data augmentation:

1. **Step #1:** An input batch of images is presented to the `ImageDataGenerator` .

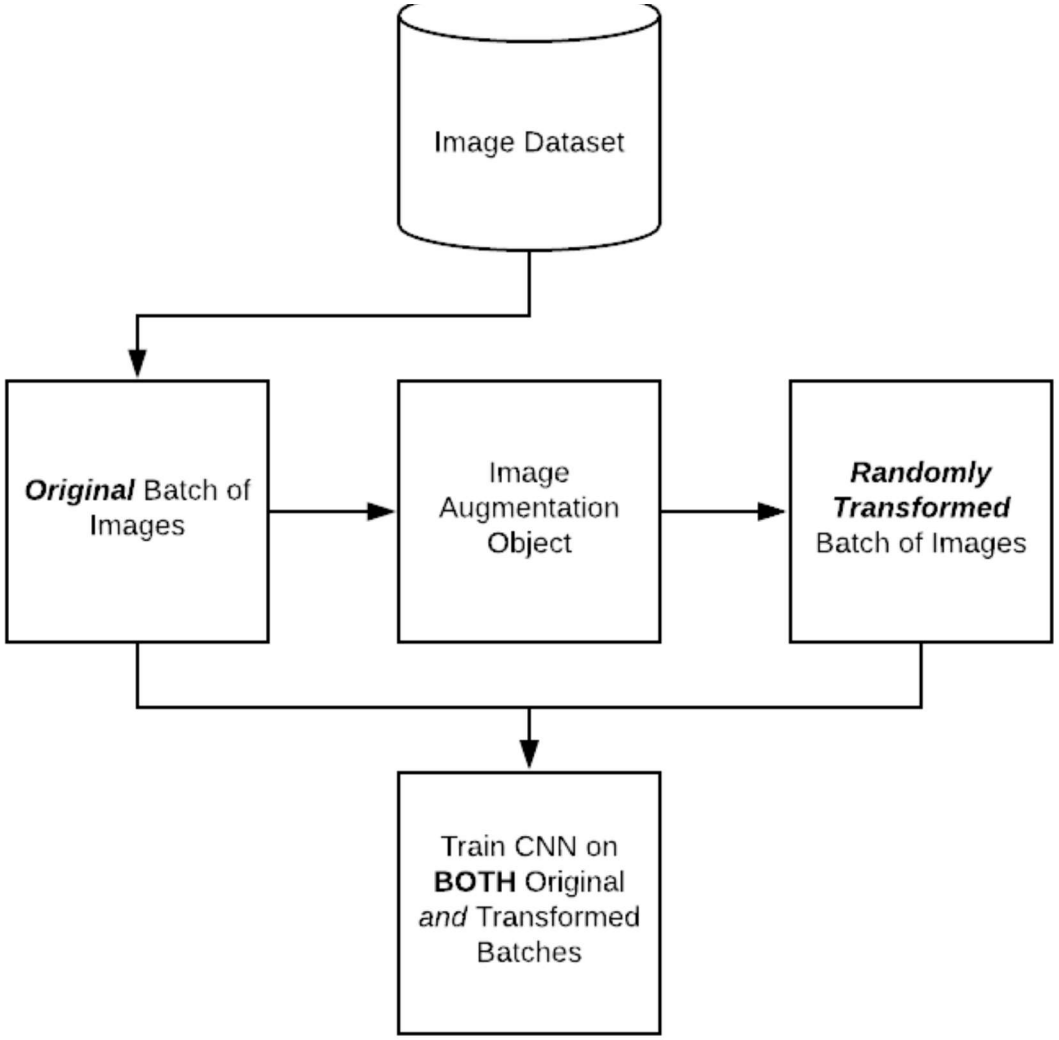
2. **Step #2:** The `ImageDataGenerator` transforms each image in the batch by a series of random translations, rotations, etc.
3. **Step #3:** The randomly transformed batch is then returned to the calling function.

There are two important points that I want to draw your attention to:

1. The `ImageDataGenerator` is *not* returning both the original data and the transformed data — ***the class only returns the randomly transformed data.***
2. We call this "in-place" and "on-the-fly" data augmentation because **this augmentation is done *at training time*** (i.e., we are not generating these examples ahead of time/prior to training).

When our model is being trained, we can think of our `ImageDataGenerator` class as "intercepting" the original data, randomly transforming it, and then returning it to the neural network for training, *all the while the NN has no idea the data was modified!*

I've written previous tutorials on the PyImageSearch blog where readers think that Keras' ImageDateGenerator class is an "additive operation", similar to the following (incorrect) figure:



(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_incorrect.png)

Figure 6: How Keras data augmentation does **not** work.

In the above illustration the `ImageDataGenerator` accepts an input batch of images, randomly transforms the batch, and then returns both the original batch and modified data — **again, this is *not* what the Keras `ImageDataGenerator` does. Instead, the `ImageDataGenerator` class will return *just* the randomly transformed data.**

When I explain this concept to readers the next question is often:

> *But Adrian, what about the original training data? Why is it not used? Isn't the original training data still useful for training?*

Keep in mind that the *entire point* of the data augmentation technique described in this section is to ensure that the network sees "new" images that it has never "seen" before *at each and every epoch*.

If we included the original training data along with the augmented data in each batch, then the network would "see" the original training data multiple times, effectively defeating the purpose. Secondly, recall that the overall goal of data augmentation is to increase the generalizability of the model.

To accomplish this goal we "replace" the training data with randomly transformed, augmented data.

In practice, this leads to a model that performs better on our validation/testing data but perhaps performs slightly worse on our training data (to due to the variations in data caused by the random transforms).

You'll learn how to use the Keras `ImageDataGenerator` class later in this tutorial.

Type #3: Combining dataset generation and in-place augmentation

The final type of data augmentation seeks to combine *both* dataset generation and in-place augmentation — you may see this type of data augmentation when performing **behavioral cloning**.

A great example of behavioral cloning can be seen in self-driving car applications.

Creating self-driving car datasets can be extremely time consuming and expensive — **a way around the issue is to instead use video games and car driving simulators.**

Video game graphics have become so life-like that it's **now possible to use them as training data (https://medium.com/@ksakmann/behavioral-cloning-make-a-car-drive-like-yourself-dc6021152713)**.

Therefore, instead of driving an *actual* vehicle, you can instead:

- Play a video game
- Write a program to play a video game
- Use the underlying rendering engine of the video game

...all to generate actual data that can be used for training.

Once you have your training data you can go back and apply Type #2 data augmentation (i.e., in-place/on-the-fly data augmentation) to the data you gathered via your simulation.

Configuring your development environment

To configure your system for this tutorial, I first recommend following either of these tutorials:

- ***How to install TensorFlow 2.0 on Ubuntu* (https://www.pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-ubuntu/)**
- ***How to install TensorFlow 2.0 on macOS (https://www.pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-macos/)***

Either tutorial will help you configure you system with all the necessary software for this blog post in a convenient Python virtual environment.

Please note that **PyImageSearch does not recommend or support Windows for CV/DL projects (https://www.pyimagesearch.com/faqs/single-faq/can-you-help-me-do-___-on-windows/)**.

Project structure

Before we dive into the code let's first review our directory structure for the project:

→ Launch Jupyter Notebook on Google Colab

```
ImageDataGenerator and Data Augmentation
$ tree --dirsfirst --filelimit 10
.
```

```
3. |  ├── dogs_vs_cats_small
4. |  |   ├── cats [1000 entries]
5. |  |   └── dogs [1000 entries]
6. |  ├── generated_dataset
7. |  |   ├── cats [100 entries]
8. |  |   └── dogs [100 entries]
9. |  ├── pyimagesearch
10. |  |   ├── __init__.py
11. |  |   └── resnet.py
12. |  ├── cat.jpg
13. |  ├── dog.jpg
14. |  ├── plot_dogs_vs_cats_no_aug.png
15. |  ├── plot_dogs_vs_cats_with_aug.png
16. |  ├── plot_generated_dataset.png
17. |  ├── train.py
18. |  └── generate_images.py
19. |
20. |  7 directories, 9 files
```

First, there are two dataset directories **which are not to be confused:**

- `dogs_vs_cats_small/` : A subset of the popular **Kaggle Dogs vs. Cats (https://www.kaggle.com/c/dogs-vs-cats)** competition dataset. In my curated subset, only 2,000 images (1,000 per class) are present (as opposed to the 25,000 images for the challenge).
- `generated_dataset/` : We'll create this generated dataset using the `cat.jpg` and `dog.jpg` images which are in the parent directory. We'll utilize data augmentation Type #1 to generate this dataset automatically and fill this directory with images.

Next, we have our `pyimagesearch` module which contains our implementation of the ResNet CNN classifier.

Today, we'll review two Python scripts:

- `train.py` : Used to train models for both **Type #1** and **Type #2** (and optionally **Type #3** if the user so wishes) data augmentation techniques. We'll perform three training experiments resulting in each of the three `plot*.png` files in the project folder.
- `generate_images.py` : Used to generate a dataset from a single image using **Type #1**.

Let's begin.

Implementing our training script

In the remainder of this tutorial we'll be performing three experiments:

1. **Experiment #1:** Generate a dataset via dataset expansion and train a CNN on it.
2. **Experiment #2:** Use a subset of the **Kaggle Dogs vs. Cats dataset (https://www.kaggle.com/c/dogs-vs-cats)** and train a CNN *without* data augmentation.
3. **Experiment #3:** Repeat the second experiment, but this time *with* data augmentation.

All of these experiments will be accomplished using the same Python script.

Open up the `train.py` script and let's get started:

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
 1. # set the matplotlib backend so figures can be saved in the background
 2. import matplotlib
 3. matplotlib.use("Agg")
 4.
 5. # import the necessary packages
 6. from pyimagesearch.resnet import ResNet
 7. from sklearn.preprocessing import LabelEncoder
 8. from sklearn.model_selection import train_test_split
 9. from sklearn.metrics import classification_report
10. from tensorflow.keras.preprocessing.image import ImageDataGenerator
11. from tensorflow.keras.optimizers import SGD
12. from tensorflow.keras.utils import to_categorical
13. from imutils import paths
14. import matplotlib.pyplot as plt
15. import numpy as np
16. import argparse
17. import cv2
18. import os
```

On **Lines 2-18** our necessary packages are imported. **Line 10** is our `ImageDataGenerator` import from the Keras library — a class for data augmentation.

Let's go ahead and **parse our command line arguments (https://pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/)**:

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
20. # construct the argument parser and parse the arguments
21. ap = argparse.ArgumentParser()
22. ap.add_argument("-d", "--dataset", required=True,
23.    help="path to input dataset")
24. ap.add_argument("-a", "--augment", type=int, default=-1,
```

```
    help="whether or not 'on the fly' data augmentation should be used")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
    help="path to output loss/accuracy plot")
args = vars(ap.parse_args())
```

Our script accepts three command line arguments via the terminal:

- `--dataset` : The path to the input dataset.
- `--augment` : Whether "on-the-fly" data augmentation should be used (refer to **type #2** above). By default, this method is *not* performed.
- `--plot` : The path to the output training history plot.

Let's proceed to initialize hyperparameters and load our image data:

→ **Launch Jupyter Notebook on Google Colab**

ImageDataGenerator and Data Augmentation

```
# initialize the initial learning rate, batch size, and number of
# epochs to train for
INIT_LR = 1e-1
BS = 8
EPOCHS = 50

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
```

```
41. | labels = []
42. |
43. | # loop over the image paths
44. | for imagePath in imagePaths:
45. |    # extract the class label from the filename, load the image, and
46. |    # resize it to be a fixed 64x64 pixels, ignoring aspect ratio
47. |    label = imagePath.split(os.path.sep)[-2]
48. |    image = cv2.imread(imagePath)
49. |    image = cv2.resize(image, (64, 64))
50. |
51. |    # update the data and labels lists, respectively
52. |    data.append(image)
53. |    labels.append(label)
```

Training hyperparameters, including initial learning rate, batch size, and number of epochs to train for, are initialized on **Lines 32-34**.

From there **Lines 39-53** grab `imagePaths`, load images, and populate our `data` and `labels` lists. The only image preprocessing we perform at this point is to resize each image to *64×64*px.

Next, let's finish preprocessing, encode our labels, and partition our data:

→ **Launch Jupyter Notebook on Google Colab**

ImageDataGenerator and Data Augmentation

```
55. | # convert the data into a NumPy array, then preprocess it by scaling
56. | # all pixel intensities to the range [0, 1]
57. | data = np.array(data, dtype="float") / 255.0
58. |
59. | # encode the labels (which are currently strings) as integers and then
60. | # one-hot encode them
61. | le = LabelEncoder()
62. | labels = le.fit_transform(labels)
```

```
63. | labels = to_categorical(labels, 2)
64. |
65. | # partition the data into training and testing splits using 75% of
66. | # the data for training and the remaining 25% for testing
67. | (trainX, testX, trainY, testY) = train_test_split(data, labels,
68. |    test_size=0.25, random_state=42)
```

On **Line 57**, we convert data to a NumPy array as well as scale all pixel intensities to the range *[0, 1]*. This completes our preprocessing.

From there we perform "one-hot encoding" of our `labels` (**Lines 61-63**). This method of encoding our `labels` results in an array that may look like this:

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
1. | array([[0., 1.],
2. |        [0., 1.],
3. |        [0., 1.],
4. |        [1., 0.],
5. |        [1., 0.],
6. |        [0., 1.],
7. |        [0., 1.]], dtype=float32)
```

For this sample of data, there are two cats (`[1., 0.]`) and five dogs (`[0., 1]`) where the label corresponding to the image is marked as "hot".

From there we partition our `data` into training and testing splits marking 75% of our data for training and the

remaining 25% for testing (**Lines 67 and 68**).

Now, we are ready to **initialize our data augmentation object:**

→ Launch Jupyter Notebook on Google Colab

```
ImageDataGenerator and Data Augmentation
70. | # initialize an our data augmenter as an "empty" image data generator
71. | aug = ImageDataGenerator()
```

Line 71 initializes our empty data augmentation object (i.e., no augmentation will be performed). This is the default operation of this script.

Let's check if we're going to override the default with the `--augment` command line argument:

→ Launch Jupyter Notebook on Google Colab

```
ImageDataGenerator and Data Augmentation
73. | # check to see if we are applying "on the fly" data augmentation, and
74. | # if so, re-instantiate the object
75. | if args["augment"] > 0:
76. |    print("[INFO] performing 'on the fly' data augmentation")
77. |    aug = ImageDataGenerator(
78. |        rotation_range=20,
79. |        zoom_range=0.15,
80. |        width_shift_range=0.2,
81. |        height_shift_range=0.2,
82. |        shear_range=0.15,
```

```
83.         horizontal_flip=True,
84.         fill_mode="nearest")
```

Line 75 checks to see if we are performing data augmentation. If so, we re-initialize the data augmentation object with random transformation parameters (**Lines 77-84**). As the parameters indicate, random rotations, zooms, shifts, shears, and flips will be performed during in-place/on-the-fly data augmentation.

Let's compile and train our model:

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
86.  # initialize the optimizer and model
87.  print("[INFO] compiling model...")
88.  opt = SGD(lr=INIT_LR, momentum=0.9, decay=INIT_LR / EPOCHS)
89.  model = ResNet.build(64, 64, 3, 2, (2, 3, 4),
90.     (32, 64, 128, 256), reg=0.0001)
91.  model.compile(loss="binary_crossentropy", optimizer=opt,
92.     metrics=["accuracy"])
93.
94.  # train the network
95.  print("[INFO] training network for {} epochs...".format(EPOCHS))
96.  H = model.fit(
97.     x=aug.flow(trainX, trainY, batch_size=BS),
98.     validation_data=(testX, testY),
99.     steps_per_epoch=len(trainX) // BS,
100.    epochs=EPOCHS)
```

2020-06-04 Update: *Formerly, TensorFlow/Keras required use of a method called* `.fit_generator` *in order to accomplish data augmentation. Now, the* `.fit` *method can handle data augmentation as well, making for more-consistent code. This also applies to the migration from* `.predict_generator` *to* `.predict`*. Be sure to check out my other article* ***fit and fit_generator (https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and***

my other article ***fit and fit_generator (https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit_generator-a-hands-on-tutorial/)*** *after you're done reading this tutorial.*

Lines 88-92 construct our `ResNet` model using Stochastic Gradient Descent optimization and learning rate decay. We use `"binary_crossentropy"` loss for this 2-class problem. If you have more than two classes, be sure to use `"categorial_crossentropy"` .

Lines 96-100 then train our model. The `aug` object handles data augmentation in batches (although be sure to recall that the `aug` object will only perform data augmentation if the `--augment` command line argument was set).

Finally, we'll evaluate our model, print statistics, and generate a training history plot:

→ **Launch Jupyter Notebook on Google Colab**

ImageDataGenerator and Data Augmentation

```
102. # evaluate the network
103. print("[INFO] evaluating network...")
104. predictions = model.predict(x=testX.astype("float32"), batch_size=BS)
105. print(classification_report(testY.argmax(axis=1),
106.    predictions.argmax(axis=1), target_names=le.classes_))
107.
108. # plot the training loss and accuracy
109. N = np.arange(0, EPOCHS)
110. plt.style.use("ggplot")
111. plt.figure()
112. plt.plot(N, H.history["loss"], label="train_loss")
113. plt.plot(N, H.history["val_loss"], label="val_loss")
114. plt.plot(N, H.history["accuracy"], label="train_acc")
115. plt.plot(N, H.history["val_accuracy"], label="val_acc")
116. plt.title("Training Loss and Accuracy on Dataset")
117. plt.xlabel("Epoch #")
118. plt.ylabel("Loss/Accuracy")
```

```
118. | plt.ylabel("Loss/Accuracy")
119. | plt.legend(loc="lower left")
120. | plt.savefig(args["plot"])
```

2020-06-04 Update: *In order for this plotting snippet to be TensorFlow 2+ compatible the* `H.history` *dictionary keys are updated to fully spell out "accuracy" sans "acc" (i.e.,* `H.history["val_accuracy"]` *and* `H.history["accuracy"]` *). It is semi-confusing that "val" is not spelled out as "validation"; we have to learn to love and live with the API and always remember that it is a work in progress that many developers around the world contribute to.*

Line 104 makes predictions on the test set for evaluation purposes. A classification report is printed via **Lines 105 and 106**.

From there, **Lines 109-120** generate and save an accuracy/loss training plot.

Generating a dataset/dataset expansion with data augmentation and Keras

In our first experiment, we will perform dataset expansion via data augmentation with Keras.

Our dataset will contain 2 classes and initially, the dataset will trivially contain only 1 image per class:

- **Cat:** 1 image
- **Dog:** 1 image

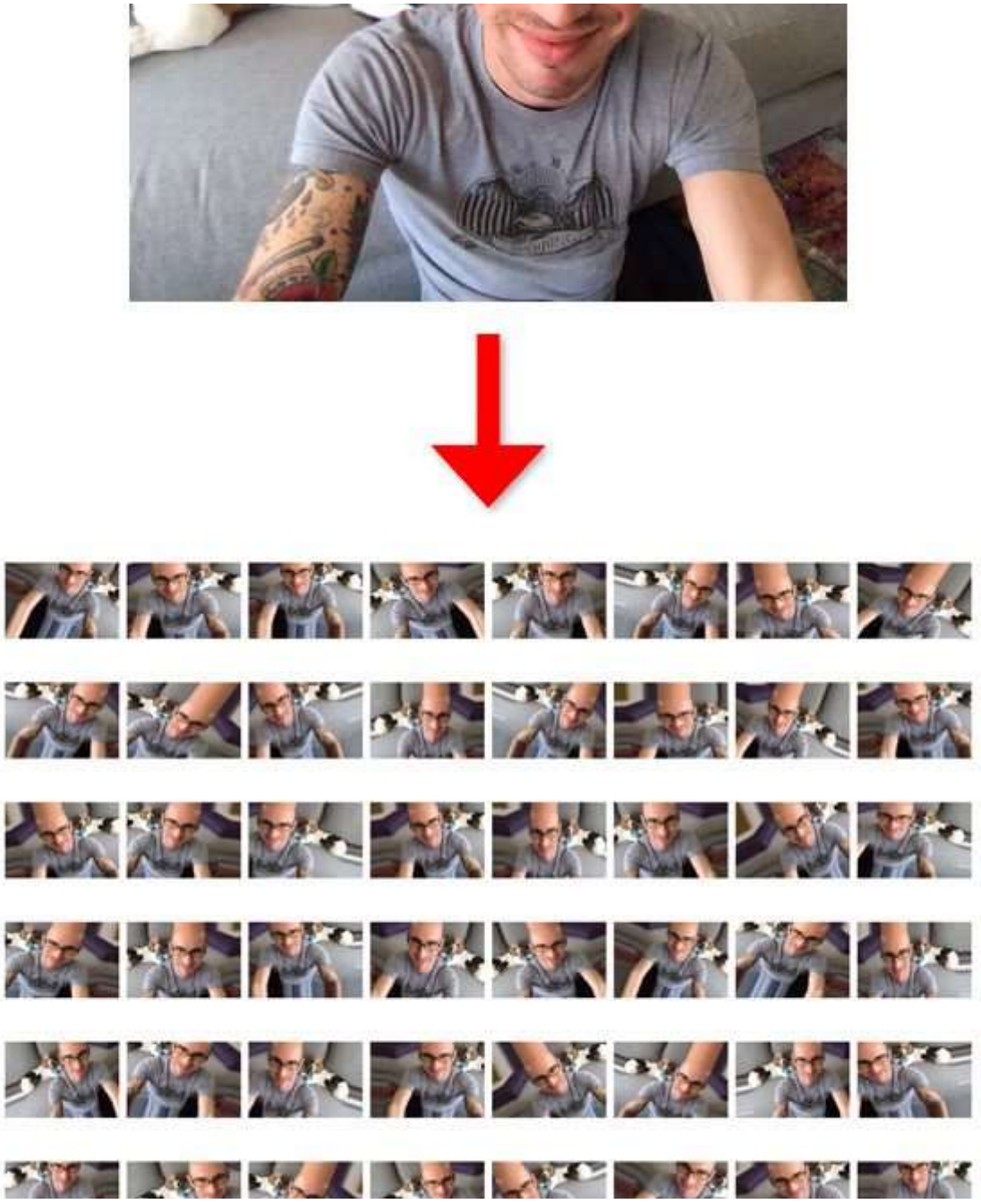
We'll utilize **Type #1** data augmentation (see the *"Type #1: Dataset generation and expanding an existing dataset"* section above) to generate a new dataset with 100 images per class:

section above) to generate a new dataset with 100 images per class:

- **Cat:** 100 images
- **Dog:** 100 images

Again, this meant to be an *example* — in a real-world application you would have 100s of example images, but we're keeping it simple here so you can learn the concept.

Generating the example dataset



(https://pyimagesearch.com/wp-content/uploads/2019/07/keras_data_augmentation_generated_dataset.jpg)

Figure 7: Data augmentation with Keras performs random manipulations on images.

Before we can train our CNN we first need to generate an example dataset.

From our *"Project Structure"* section above you know that we have two example images in our root directory: `cat.jpg` and `dog.jpg`. We will use these example images to generate 100 new training images per class (200 images in total).

To see how we can use data augmentation to generate new examples, open up the `generate_images.py` file and follow along:

→ **Launch Jupyter Notebook on Google Colab**

→ **Launch Jupyter Notebook on Google Colab**

ImageDataGenerator and Data Augmentation

```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
import numpy as np
import argparse

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
```

```
10. | ap.add_argument("-i", "--image", required=True,
11. |    help="path to the input image")
12. | ap.add_argument("-o", "--output", required=True,
13. |    help="path to output directory to store augmentation examples")
14. | ap.add_argument("-t", "--total", type=int, default=100,
15. |    help="# of training samples to generate")
16. | args = vars(ap.parse_args())
```

Lines 2-6 import our necessary packages. Our `ImageDataGenerator` is imported on **Line 2** and will handle our data augmentation with Keras.

From there, we'll parse three command line arguments:

- `--image` : The path to the input image. We'll generate additional random, mutated versions of this image.
- `--output` : The path to the output directory to store the data augmentation examples.
- `--total` : The number of sample images to generate.

Let's go ahead and load our `image` and **initialize our data augmentation object:**

→ Launch Jupyter Notebook on Google Colab

```
ImageDataGenerator and Data Augmentation

18. | # load the input image, convert it to a NumPy array, and then
19. | # reshape it to have an extra dimension
20. | print("[INFO] loading example image...")
21. | image = load_img(args["image"])
22. | image = img_to_array(image)
23. | image = np.expand_dims(image, axis=0)
```

```
24. |
25. |  # construct the image generator for data augmentation then
26. |  # initialize the total number of images generated thus far
27. |  aug = ImageDataGenerator(
28. |     rotation_range=30,
29. |     zoom_range=0.15,
30. |     width_shift_range=0.2,
31. |     height_shift_range=0.2,
32. |     shear_range=0.15,
33. |     horizontal_flip=True,
34. |     fill_mode="nearest")
35. |  total = 0
```

Our `image` is loaded and prepared for data augmentation via **Lines 21-23**. Image loading and processing is handled via Keras functionality (i.e. we aren't using OpenCV).

From there, we initialize the `ImageDataGenerator` object. This object will facilitate performing random rotations, zooms, shifts, shears, and flips on our input image.

Next, we'll construct a Python generator and put it to work until all of our images have been produced:

→ **Launch Jupyter Notebook on Google Colab**

ImageDataGenerator and Data Augmentation

```
37. |  # construct the actual Python generator
38. |  print("[INFO] generating images...")
39. |  imageGen = aug.flow(image, batch_size=1, save_to_dir=args["output"],
40. |     save_prefix="image", save_format="jpg")
41. |
42. |  # loop over examples from our image data augmentation generator
43. |  for image in imageGen:
44. |     # increment our counter
45. |     total += 1
```

```
46. |
47. |     # if we have reached the specified number of examples, break
48. |     # from the loop
49. |     if total == args["total"]:
50. |         break
```

We will use the `imageGen` to randomly transform the input image (**Lines 39 and 40**). This generator saves images as .jpg files to the specified output directory contained within `args["output"]` .

Finally, we'll loop over examples from our image data generator and count them until we've reached the required `total` number of images.

To run the `generate_examples.py` script make sure you have used the ***"Downloads"*** section of the tutorial to download the source code and example images.

From there open up a terminal and execute the following command:

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
1. |  $ python generate_images.py --image cat.jpg --output generated_dataset/cats
2. |  [INFO] loading example image...
3. |  [INFO] generating images...
```

Check the output of the `generated_dataset/cats` directory you will now see 100 images:

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
$ ls generated_dataset/cats/*.jpg | wc -l
      100
```

Let's do the same now for the "dogs" class:

→ **Launch Jupyter Notebook on Google Colab**

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
$ python generate_images.py --image dog.jpg --output generated_dataset/dogs
[INFO] loading example image...
[INFO] generating images...
```

And now check for the dog images:

→ **Launch Jupyter Notebook on Google Colab**

→ **Launch Jupyter Notebook on Google Colab**

```
ImageDataGenerator and Data Augmentation
$ ls generated_dataset/dogs/*.jpg | wc -l
      100
```

A visualization of the dataset generation via data augmentation can be seen in **Figure 6** at the top of this section — notice how we have accepted a single input image (of me — not of a dog or cat) and then created 100 new training examples (48 of which are visualized) from that single image.

Experiment #1: Dataset generation results

We are now ready to perform our first experiment:

→ **Launch Jupyter Notebook on Google Colab**

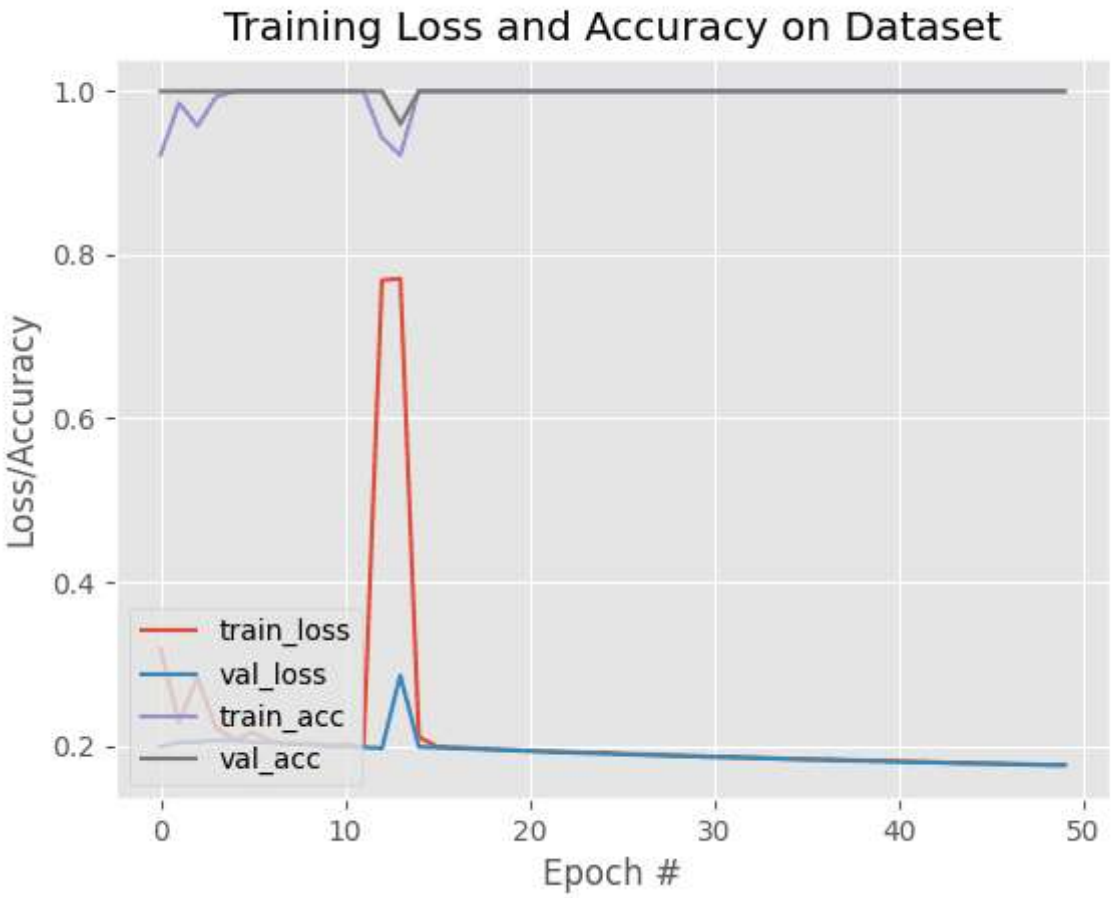
→ **Launch Jupyter Notebook on Google Colab**

ImageDataGenerator and Data Augmentation

```
$ python train.py --dataset generated_dataset --plot plot_generated_dataset.png
[INFO] loading images...
[INFO] compiling model...
[INFO] training network for 50 epochs...
Epoch 1/50
18/18 [==============================] - 1s 60ms/step - loss: 0.3191 - accuracy: 0.9220 - val_loss: 0.1986 - val_accuracy:
1.0000
Epoch 2/50
18/18 [==============================] - 0s 9ms/step - loss: 0.2276 - accuracy: 0.9858 - val_loss: 0.2044 - val_accuracy:
1.0000
Epoch 3/50
18/18 [==============================] - 0s 8ms/step - loss: 0.2839 - accuracy: 0.9574 - val_loss: 0.2046 - val_accuracy:
1.0000
...
Epoch 48/50
18/18 [==============================] - 0s 9ms/step - loss: 0.1770 - accuracy: 1.0000 - val_loss: 0.1768 - val_accuracy:
1.0000
Epoch 49/50
18/18 [==============================] - 0s 9ms/step - loss: 0.1767 - accuracy: 1.0000 - val_loss: 0.1763 - val_accuracy:
1.0000
Epoch 50/50
18/18 [==============================] - 0s 8ms/step - loss: 0.1767 - accuracy: 1.0000 - val_loss: 0.1758 - val_accuracy:
1.0000
[INFO] evaluating network...
              precision    recall  f1-score   support

        cats       1.00      1.00      1.00        25
        dogs       1.00      1.00      1.00        25
```

```
23. |
24. |      accuracy                           1.00         50
25. |     macro avg       1.00      1.00      1.00         50
26. |  weighted avg       1.00      1.00      1.00         50
```



(https://www.pyimagesearch.com/wp-content/uploads/2019/07/plot_generated_dataset.png)

Figure 8: Data augmentation with Keras **Experiment #1** training accuracy/loss results.

Our results show that we were able to obtain **100%** accuracy with little effort.

Of course, this is a trivial, contrived example. In practice, you would not be taking only a *single* image and then building a dataset of 100s or 1000s of images via data augmentation. Instead, you would have a dataset of 100s of images and then you would apply dataset generation to that dataset — but again, the point of this section was to demonstrate on a simple example so you could understand the process.

Training a network with in-place data augmentation

The more popular form of (image-based) data augmentation is called **in-place data augmentation** (see the *"Type #2: In-place/on-the-fly data augmentation"* section of this post for more details).

When performing in-place augmentation our Keras `ImageDataGenerator` will:

1. Accept a batch of input images.
2. Randomly transform the input batch.
3. Return the transformed batch to the network for training.

We'll explore how data augmentation can reduce overfitting and increase the ability of our model to generalize via two experiments.

To accomplish this task we'll be using a subset of the **Kaggle Dogs vs. Cats dataset**

(https://www.kaggle.com/c/dogs-vs-cats):

- **Cats:** 1,000 images
- **Dogs:** 1,000 images

We'll then train a variation of ResNet, from scratch, on this dataset with and without data augmentation.

Experiment #2: Obtaining a baseline (no data augmentation)

In our first experiment we'll perform no data augmentation:

→ Launch Jupyter Notebook on Google Colab

→ Launch Jupyter Notebook on Google Colab

```
ImageDataGenerator and Data Augmentation
$ python train.py --dataset dogs_vs_cats_small --plot plot_dogs_vs_cats_no_aug.png
[INFO] loading images...
[INFO] compiling model...
[INFO] training network for 50 epochs...
Epoch 1/50
187/187 [==============================] - 3s 13ms/step - loss: 1.0743 - accuracy: 0.5134 - val_loss: 0.9116 - val_accuracy: 0.5440
Epoch 2/50
187/187 [==============================] - 2s 9ms/step - loss: 0.9149 - accuracy: 0.5349 - val_loss: 0.9055 - val_accuracy: 0.4940
Epoch 3/50
187/187 [==============================] - 2s 9ms/step - loss: 0.9065 - accuracy: 0.5409 - val_loss: 0.8990 - val_accuracy: 0.5360
...
Epoch 48/50
187/187 [==============================] - 2s 9ms/step - loss: 0.2796 - accuracy: 0.9564 - val_loss: 1.4528 - val_accuracy:
```

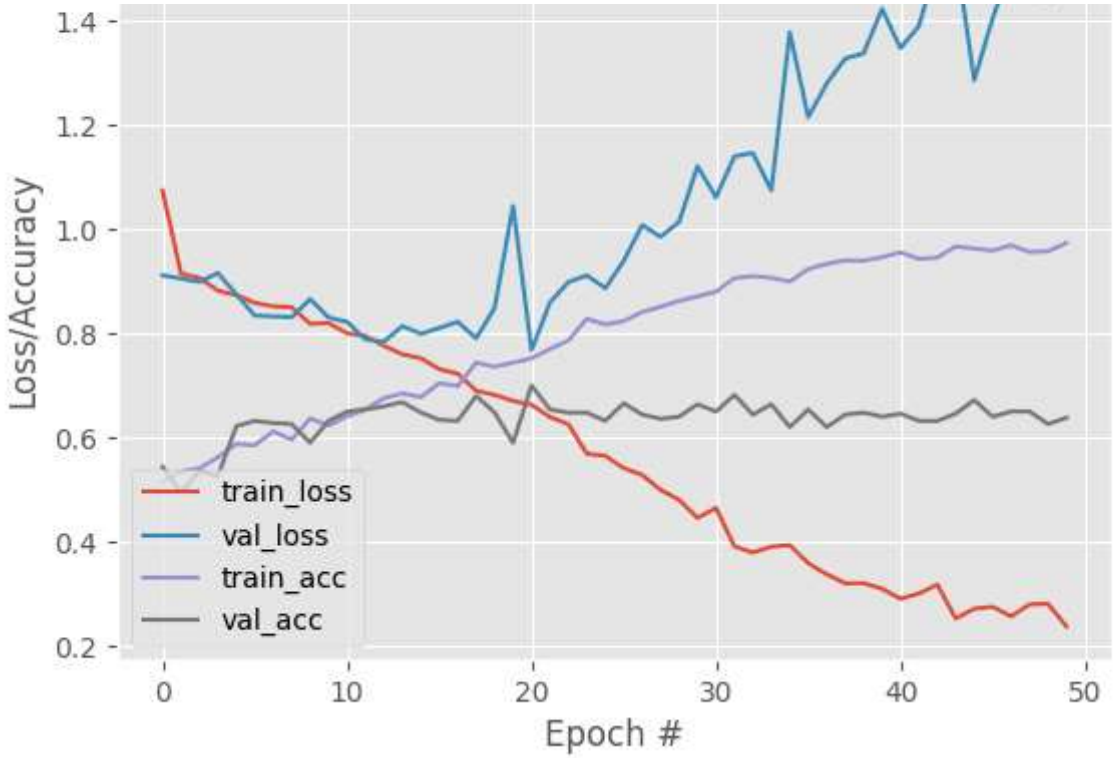
```
187/187 [==============================] - 2s 9ms/step - loss: 0.2796 - accuracy: 0.9504 - val_loss: 1.4526 - val_accuracy:
0.6500
Epoch 49/50
187/187 [==============================] - 2s 10ms/step - loss: 0.2806 - accuracy: 0.9578 - val_loss: 1.4485 - val_accuracy:
0.6260
Epoch 50/50
187/187 [==============================] - 2s 9ms/step - loss: 0.2371 - accuracy: 0.9739 - val_loss: 1.5061 - val_accuracy:
0.6380
[INFO] evaluating network...
              precision    recall  f1-score   support

        cats       0.61      0.70      0.65       243
        dogs       0.67      0.58      0.62       257

    accuracy                           0.64       500
   macro avg       0.64      0.64      0.64       500
weighted avg       0.64      0.64      0.64       500
```

Looking at the raw classification report you'll see that we're obtaining **64% accuracy — but there's a problem!**

Take a look at the plot associated with our training:



(https://www.pyimagesearch.com/wp-content/uploads/2019/07/plot_dogs_vs_cats_no_aug.png)

Figure 9: For **Experiment #2** we did not perform data augmentation. The result is a plot with strong indications of overfitting.

There is *dramatic overfitting* occurring — at approximately epoch 15 we see our validation loss start to rise while training loss continues to fall. By epoch 20 the rise in validation loss is especially pronounced.

This type of behavior is indicative of overfitting.

The solution is to (1) reduce model capacity, and/or (2) perform regularization.

Experiment #3: Improving our results (with data augmentation)

Let's now investigate how data augmentation can act as a form of regularization:

→ **Launch Jupyter Notebook on Google Colab**

→ **Launch Jupyter Notebook on Google Colab**

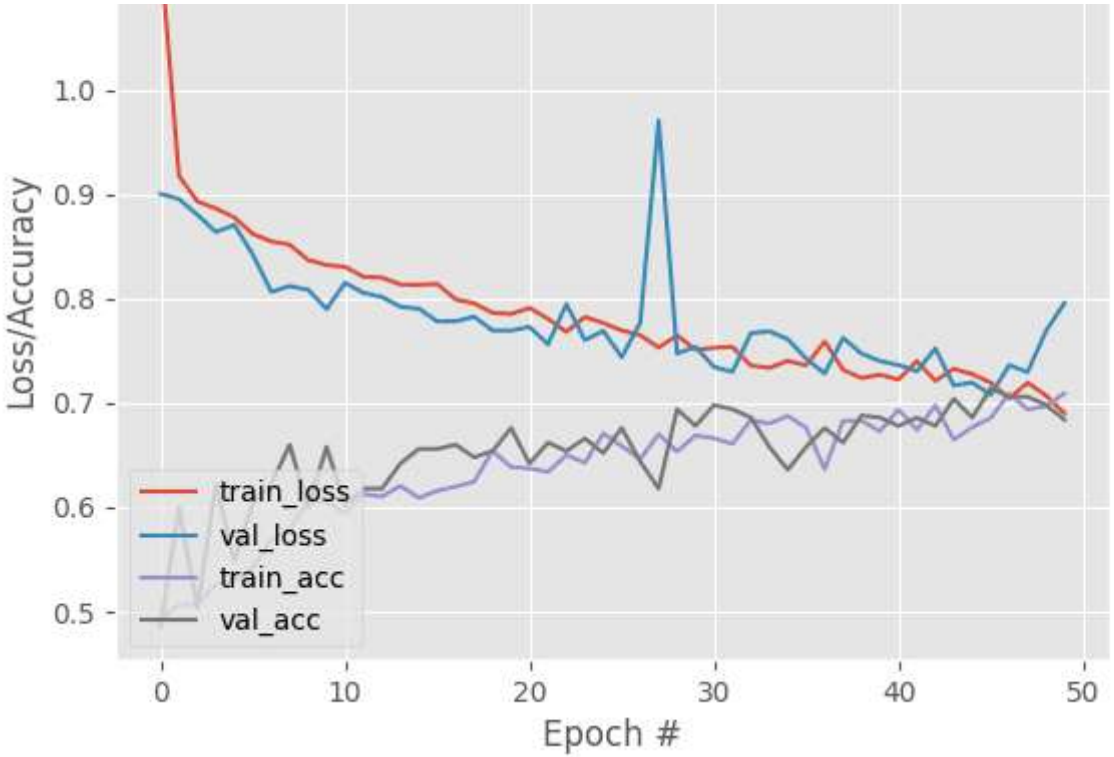
```
ImageDataGenerator and Data Augmentation
[INFO] loading images...
[INFO] performing 'on the fly' data augmentation
[INFO] compiling model...
[INFO] training network for 50 epochs...
Epoch 1/50
187/187 [==============================] - 3s 14ms/step - loss: 1.1307 - accuracy: 0.4940 - val_loss: 0.9002 - val_accuracy: 0.4860
Epoch 2/50
187/187 [==============================] - 2s 8ms/step - loss: 0.9172 - accuracy: 0.5067 - val_loss: 0.8952 - val_accuracy: 0.6000
Epoch 3/50
187/187 [==============================] - 2s 8ms/step - loss: 0.8930 - accuracy: 0.5074 - val_loss: 0.8801 - val_accuracy: 0.5040
...
Epoch 48/50
187/187 [==============================] - 2s 8ms/step - loss: 0.7194 - accuracy: 0.6937 - val_loss: 0.7296 - val_accuracy: 0.7060
Epoch 49/50
187/187 [==============================] - 2s 8ms/step - loss: 0.7071 - accuracy: 0.6971 - val_loss: 0.7690 - val_accuracy: 0.6980
Epoch 50/50
187/187 [==============================] - 2s 9ms/step - loss: 0.6901 - accuracy: 0.7091 - val_loss: 0.7957 - val_accuracy: 0.6840
[INFO] evaluating network...
              precision    recall  f1-score   support

        cats       0.72      0.56      0.63       243
        dogs       0.66      0.80      0.72       257
```

```
24. |      accuracy                           0.68       500
25. |     macro avg       0.69      0.68      0.68       500
26. |  weighted avg       0.69      0.68      0.68       500
```

We're now up to **69% accuracy**, an increase from our previous 64% accuracy.

But more importantly, we are no longer overfitting:



(https://www.pyimagesearch.com/wp-content/uploads/2019/07/plot_dogs_vs_cats_with_aug.png)

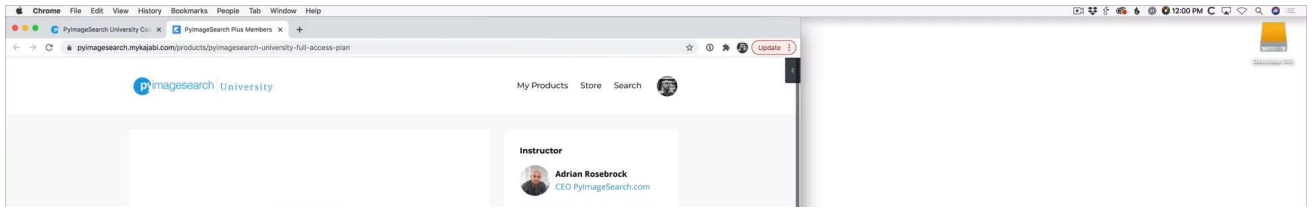
Figure 10: For **Experiment #3**, we performed data augmentation with Keras on batches of images in-place. Our training plot shows no signs of overfitting with this form of regularization.

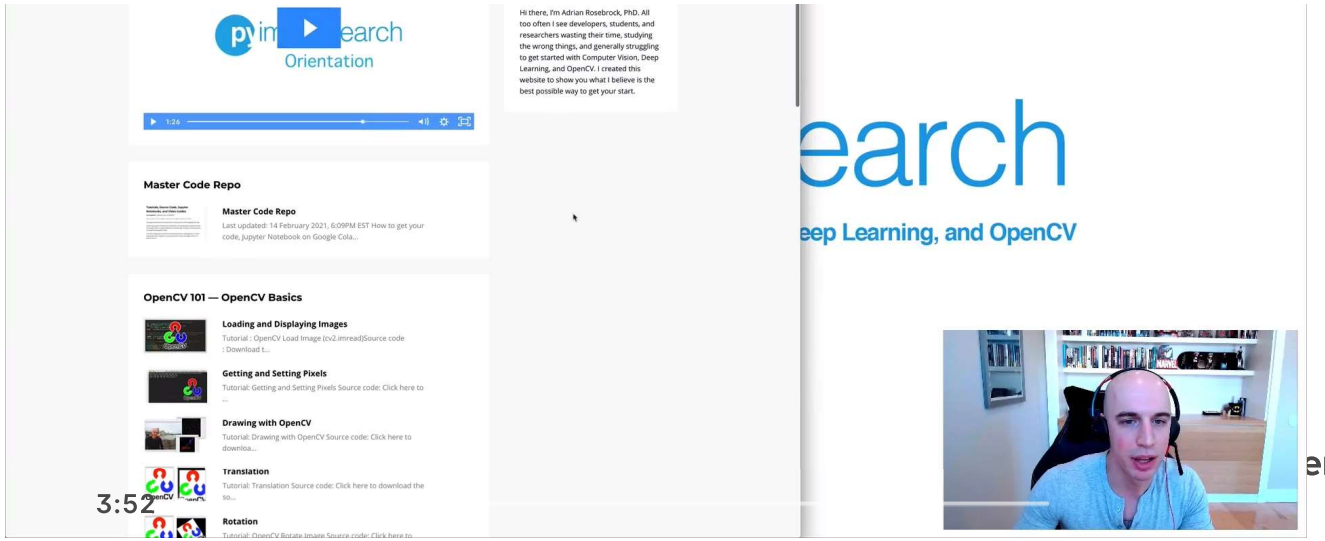
Note how validation and training loss are falling together with little divergence. Similarly, classification accuracy for both the training and validation splits are growing together as well.

By using data augmentation we were able to combat overfitting!

In nearly all situations, unless you have very good reason not to, **you should be performing data augmentation when training your own neural networks.**

What's next? I recommend PyImageSearch University (https://www.pyimagesearch.com/pyimagesearch-university/?utm_source=blogPost&utm_medium=bottomBanner&utm_campaign=What%27s%20next%3F%20I%20recommend).





er vision and deep learning.

Do you think learning computer vision and deep learning has to be time-consuming, overwhelming, and complicated? Or has to involve complex mathematics and equations? Or requires a degree in computer science?

That's *not* the case.

All you need to master computer vision and deep learning is for someone to explain things to you in *simple, intuitive* terms. *And that's exactly what I do*. My mission is to change education and how complex Artificial Intelligence topics are taught.

If you're serious about learning computer vision, your next stop should be PyImageSearch University, the most comprehensive computer vision, deep learning, and OpenCV course online today. Here you'll learn how to *successfully* and *confidently* apply computer vision to your work, research, and projects. Join me in computer vision mastery.

computer vision mastery.

Inside PyImageSearch University you'll find:

✓ **28 courses** on essential computer vision, deep learning, and OpenCV topics

✓ 28 Certificates of Completion

✓ **39h 44m** on-demand video

✓ **Brand new courses released *every month***, ensuring you can keep up with state-of-the-art techniques

✓ **Pre-configured Jupyter Notebooks in Google Colab**

✓ Run all code examples in your web browser — works on Windows, macOS, and Linux (no dev environment configuration required!)

✓ Access to **centralized code repos for *all* 400+ tutorials** on PyImageSearch

✓ **Easy one-click downloads** for code, datasets, pre-trained models, etc.

✓ Access on mobile, laptop, desktop, etc.

CLICK HERE TO JOIN PYIMAGESEARCH UNIVERSITY

(HTTPS://WWW.PYIMAGESEARCH.COM/PYIMAGESEARCH-UNIVERSITY/?UTM_SOURCE=BLOGPOST&UTM_MEDIUM=BOTTOMBANNER&UTM_CAMPAIGN=WHAT%27S%20NEXT%3F%20I%20RECOMMEND)

F%20I%20RECOMMEND)

Summary

In this tutorial, you learned about data augmentation and how to apply data augmentation via Keras' ImageDataGenerator class.

You also learned about three types of data augmentation, including:

1. Dataset generation and data expansion via data augmentation (less common).
2. In-place/on-the-fly data augmentation (most common).
3. Combining the dataset generator and in-place augmentation.

By default, **Keras' `ImageDataGenerator` class performs in-place/on-the-fly data augmentation**, meaning that the class:

1. Accepts a batch of images used for training.
2. Takes this batch and applies a series of random transformations to each image in the batch.

3. Replaces the original batch with the new, randomly transformed batch
4. 4. Trains the CNN on this randomly transformed batch (i.e., the original data itself is *not* used for training).

All that said, we actually can take the `ImageDataGenerator` class and use it for dataset generation/expansion as well — we just need to use it to generate our dataset *before* training.

The final method of data augmentation, combining both in-place and dataset expansion, is rarely used. In those situations, you likely have a small dataset, need to generate additional examples via data augmentation, and then have an additional augmentation/preprocessing at training time.

We wrapped up the guide by performing a number of experiments with data augmentation, **noting that data augmentation is a form of *regularization***, enabling our network to generalize better to our testing/validation set.

This claim of data augmentation as regularization was verified in our experiments when we found that:

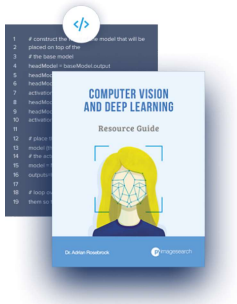
1. Not applying data augmentation at training caused overfitting
2. While apply data augmentation allowed for smooth training, no overfitting, and higher accuracy/lower loss

You should apply data augmentation in all of your experiments unless you have a very good reason not to.

To learn more about data augmentation, including my best practices, tips, and suggestions, be sure to take a look at my book, ***Deep Learning for Computer Vision with Python*** **(https://pyimagesearch.com/deep-learning-computer-vision-python-book/)**.

I hope you enjoyed today's tutorial!

To download the source code to this post (and receive email updates when future tutorials are published here on PyImageSearch), *just enter your email address in the form below!*



Download the Source Code and FREE 17-page Resource Guide

Enter your email address below to get a .zip of the code and a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL!

About the Author

Hi there, I'm Adrian Rosebrock, PhD. All too often I see developers, students, and

Hi there, I'm Adrian Rosebrock, PhD. All too often I see developers, students, and researchers wasting their time, studying the wrong things, and generally struggling to get started with Computer Vision, Deep Learning, and OpenCV. I created this website to show you what I believe is the best possible way to get your start.

Previous Article:

Remote development on the Raspberry Pi (or Amazon EC2)

(https://www.pyimagesearch.com/2019/07/01/remote-development-on-the-raspberry-pi-or-amazon-ec2/)

Next Article:

Video classification with Keras and Deep Learning

(https://www.pyimagesearch.com/2019/07/15/video-classification-with-keras-and-deep-learning/)

53 responses to: Keras ImageDataGenerator and Data Augmentation

Nikesh

July 8, 2019 at 11:36 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-524914)

Hi Adrian,

This article is awesome. It was a great selection of topic by you and the twitter poll says it all that most of the people were in misconception including me that's why it totally astonished me. But at the end I have a couple of questions:

For keras ImageDataGenerator() class, if we provide a batch of size 32 (images) from original dataset and suppose it apply 5 transformations per image then
1. Will it train on 160 images for that batch? If yes, then what happens in subsequent epochs?
2. Does on-the-fly data augmentation discards the generated images after training?

Adrian Rosebrock
July 8, 2019 at 12:06 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-524921)

Hey Nikesh,

1. you should go back and **re-read** the "Type #2: In-place/on-the-fly data augmentation (most common)" section. If you use the ImageDataGenerator class with a batch size of 32, you'll put 32 images into the object and get 32 randomly transformed images back out. You will NOT have 160 images. The same size batch comes out, only with the randomly transformed images.

the randomly transformed images.

2. Yes, all data augmentation operations are done in memory so the generated images are discarded.

Nikesh

July 9, 2019 at 9:58 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525061)

Ok so the ImageDataGenerator() object returns 32 images by applying random transformations which is used to train. In the next epoch there are 32 differently transformed images for the same batch. Am I getting it correct?

Does this introduces some kind of regularization too?

Adrian Rosebrock

July 10, 2019 at 8:12 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525163)

1. Your understanding is correct.
2. See the post. I mention multiple times how image augmentation IS a form of regularization.

Peter Shanks (https://www.csiro.au/en/Research/Facilities/Marine-National-Facility/RV-Investigator)
July 8, 2019 at 7:36 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-524976)

Hi Adrian,

I've been dabbling with CNNs for a while now, and really enjoy reading your posts.

Recently I've started building an image classifier for a collection of plankton images that we have here where I work, and it struck me that my data could be made more useful with another form of augmentation: defining where the joints and shell sections of the creatures are (and possibly how many degrees of rotation might be allowed for each joint) and then generating new images by randomly manipulating the exoskeleton.

Have you come across anything that does this or am I better off doing a crash course in object detection and Blender scripting?

Adrian Rosebrock

July 10, 2019 at 8:11 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525162)

You might want to take a look at the **imgaug library. (https://github.com/aleju/imgaug)** It provides WAY more

options for augmentation than Keras' standard ImageDataGenerator class — it might have something in the implementation that will help you.

Rémi (https://github.com/remi2257)
July 9, 2019 at 3:58 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525028)

Hey Adrian,

As usual, very good tutorial. I was among people who didn't know that the original batch is not used after data augmentation … Thx for the information !

I was wondering: is there a way to display the new batch after data augmentation ? Like:

for img,label in augmented_data:
cv2.imshow("XXX", img)
cv2.waitKey()

It could be an interesting point, for instance, when I rescale MNIST dataset, I don't want to zoom too much and generate inoperable images !

Have a good day ☺

Have a good day

Adrian Rosebrock

July 10, 2019 at 8:10 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525161)

You don't get to see the batch before it's passed into the network for training. Displaying the image would wouldn't help training and it would only slow it down due to the I/O operations. If you want to visualize the output of data augmentation see the `generate_examples.py` script.

Rémi (https://github.com/remi2257)

July 11, 2019 at 8:39 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525326)

Adrian,

I think that you did not understand my question at first time ^^. I do not want to see batches during training but before launching my training, so that I will be able to say "Ok Rémi, digits are over-cropped, you put scale parameter too high, it won't be pertinent to send that type of "over-zoomed" pictures to your CNN"
Your second answer is what I was expecting, I will check out !

Thanks

Gautam Kumar

July 10, 2019 at 7:26 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525155)

Hi Adrian,
Thanks for explaining the data augmentation clearly. Even, when I answered your question, I found myself in the category of 65% of people. But now I got your point. Well, I was working on iris recognition system using CNN with a limited number of images in each class (5 to 10) and I was really confused whether I should use data augmentation or not, as iris images comes with close capture of the ocular region and use of data augmentation will translate and Shear the eye shape. Can you please suggest whether I should use data augmentation for iris recognition or not?

Adrian Rosebrock

July 10, 2019 at 8:13 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525164)

With only 5-10 example images you should instead invest your time gathering additional training data instead of trying to perform data augmentation. 5-10 images realistically isn't enough.

Gautam Kumar

July 11, 2019 at 2:51 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525280)

Thanks for your quick reply, actually I am using UBIRIS.v1 which is well known and standard dataset. I can't add images in this database. Well, I wanted to know, if I perform data augmentation on these images, it would change/reshape image due to the random transformation that may cause the iris region (which is important for my experiment) may get cropped or partially available for feature extraction. It would be helpful for me if you suggest or write blogs stating in which case and with what type of images data augmentation is a helpful approach for training the network and where it should not be used. Thanks

Nurhan

July 10, 2019 at 2:33 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525230)

Hi Adrian,

thank you very much for this nice tutorial!

I have two questions:

1. does the ImageDataGenerator also augment the annotations of images? for example the masks of Mask-RCNN in JSON?

2. is the imgaug library able to generate as well augmented images and save them on the fly?

Adrian Rosebrock

July 15, 2019 at 1:12 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525787)

No, you would want to use **imgaug (https://github.com/aleju/imgaug)** for annotation of object detection, instance segmentation, etc.

Denys

July 11, 2019 at 10:10 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525345)

Hi Adrian, thanks for this article. I wish I saw this article before when I was starting ML.
I was developing a mobile app Frutolo. It is for the recognition of fruits and vegetables.
I was really limited about the number of training images. So, that's how I started with image augmentation.
It is especially important for such applications, where the shapes and colours of the objects are very similar.

It is especially important for such applications, where the shapes and colours of the objects are very similar.

Adrian Rosebrock

July 15, 2019 at 1:11 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525786)

Thanks Denys, I'm glad you enjoyed the article. And congrats on building your DL application ☺

Patrick Ryan

July 11, 2019 at 8:41 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525389)

Hi Adrian
Thank you for this great article and all of your articles. I am big fan of your work and I have read the python and opencv book – excellent!, I am working through your Deep Learning Book which is fantastic! and I signed up for the Raspberry PI kickstarter.

My question is whether image data generation and augmentation makes sense when creating a training set for facial recognition. If I understand correctly, facial recognition tries to normalize a face (straight on, no rotation, etc) so if I create an augmented set will the facial recognition software just try to undo the small transformations?

I did try it with my family members LinkedIn pictures. I took the one profile picture from LinkedIn, generated 30 from the one picture, then used the techniques you describe in this post:

https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/ (https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/)

and the face_recognition package was able to recognize each person. I know you are busy, thank you for taking time. I would love your thoughts on this approach.

Again – thank you.

Adrian Rosebrock

July 15, 2019 at 1:10 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525785)

It won't undo it per se, but it will re-align the face. It's still valuable to apply data augmentation to face recognition.

M Sudhakara

July 12, 2019 at 1:19 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525410)

Hi Adrian,

Can I use Color transformations as one of data augmentation? For example, I have RGB color image. If I convert this RGB to HSV, LAB we get the same image in other color spaces.

Thes converted images can act as a data augmentation?

Adrian Rosebrock

July 15, 2019 at 1:09 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525784)

You typically wouldn't change the color space and use JUST the modified color space for your training data. You pick a color space for training and stick with that color space. You may decide to adjust contrast, brightness, etc. For that take a look at the **imgaug library. (https://github.com/aleju/imgaug)**

monty

July 12, 2019 at 2:45 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525414)

why doesn't such a big network identify that images are linear combination of each other? or images are derived

from other images.

Xu Zhang

July 12, 2019 at 6:42 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525491)

Thank you so much for your great post. Finally, I understood how augmentation works.
For Keras ImageDataGenerator(), it can do in-place/on-the-fly data augmentation for 2D image with 3 channels. Can it do 3D with more than 3 channels augmentation on the fly? If not, whether do there some implementations exist?
Many thanks

Adrian Rosebrock

July 15, 2019 at 1:08 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525783)

I'm not sure about that one. You may want to look into the "imgaug" library. That library might be able to help you out more.

asere

August 6, 2019 at 10:55 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-532005)

Also interested in this, were you able to find any existing implementations or did dataug library work for you in 3D images?

KV Subbaiah Setty

July 12, 2019 at 9:03 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525503)

Hello Adrian,

I am thinking of a trick ,since in ” #type two” data augmentation the original images dataset is not seen by the model during training (all train examples are transformed images) , can we use the original images (original train set) as validation data set and there is no need of train- test split ,as all all original images are unseen images during training. What I am thinking is something like “out-of-bag” samples (oob sampling) usually done in bagging .

What is your opinion and thoughts on this ? Is my thinking is correct?.

Adrian Rosebrock

July 15, 2019 at 1:07 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525782)

No, don’t do that. Keep your training, testing, and validation sets entirely independent and sequestered from

each other. Trying to mix them could increase the chance of overfitting and if you were to publish a paper it could even invalidate your results. Keep them separate.

Nurhan

July 13, 2019 at 11:20 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525571)

Hey Adrian,

How to use data augmentation with YOLO and SSD? Where to put the data augmentation there?

Adrian Rosebrock

July 15, 2019 at 1:06 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-525781)

Take a look at the **imgaug library. (https://github.com/aleju/imgaug)**

Future_Vision

July 22, 2019 at 6:24 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-526792)

Hi Adrian,
I'm really happy I found your blog. Much of what I am trying to accomplish you cover but I am running into an issue with data augmentation. I'm just not sure what the right approach is. Here is my high level explanation of my project. I have a set of 52 playing cards of a specific design. Since finding images of this design is very difficult my thinking was to generate images for each of the 52 cards(classes). I have 3 directories(train, validate, test) and each of those there is a directory for each card. I have an image of the original, unadulterated, art for each card sitting in each of those directories. I've been able to generate images for a single image but only certain transformations work but that is probably another question. Just trying to figure out how to batch generate images for all 52 classes without manually doing it for each class in each set(train, validate and test). Any guidance you can give me? I'd greatly appreciate it!

Adrian Rosebrock

July 25, 2019 at 9:26 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-527236)

Hey there — have you taken a look at **Deep Learning for Computer Vision with Python? (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)** That book covers my tips, suggestions, and best practices for how to build your own datasets and generators. All of my guidance and suggestions are covered in that book. Definitely take a look if you're serious about your project.

suggestions are covered in that book. Definitely take a look if you're serious about your project.

Future_Vision

July 25, 2019 at 10:10 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-527294)

Thanks for the feedback. This is just a class project I am a little stumped on so I'll need to pass on buying the book.

Adrian Rosebrock

July 25, 2019 at 10:20 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-527309)

Good luck with the class project! I hope it turns out well.

Joe

July 26, 2019 at 11:16 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-527449)

Hi Adrian,

Well, I don't need to say how important this blog has been to my short ML life. Thank you.

I've got a question, a dummy question, you know.
I am new in python. I wonder if you could me explain this part:
label = imagePath.split(os.path.sep)[-2]

Thank you

Walid

August 2, 2019 at 10:40 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-529960)

Thanks a lot. I appreciate all the effory you put in this post

I have a question not related directly to the post, Why you always prefer to have a last layer with 2 neurons while you are doing binary classiffier.?
Would not a single neuron do the trick?

Best Reagrds,

Walid

Sammy

August 9, 2019 at 4:23 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-533833)

Hi Adrian,

What code could you use if you wanted to include both the original images and the augmented images in training?

Adrian Rosebrock

August 16, 2019 at 6:00 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-539246)

You would need to implement **your own custom data generator (https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit_generator-a-hands-on-tutorial/)** and inside the generator have it apply data augmentation to the images. Then, concatenate the original images with the augmented images and yield it.

with the augmented images and yield it.

Pankaj

August 16, 2019 at 6:46 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-539274)

Hey Adrain,

I am fan your style, Now I have started following you through this blog.

Have a look at this thread: **https://www.kaggle.com/questions-and-answers/96043#latest-555722 (https://www.kaggle.com/questions-and-answers/96043#latest-555722)**

Adrian Rosebrock

September 5, 2019 at 9:59 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-547063)

Thanks Pankaj, I'm happy the tutorial helped you ☺

Ashish

September 4, 2019 at 2:32 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-546752)

Hi Adrian,
Can you please help how we can implement “ImageDataGenerator” and “Data Augmentation” on multi-label image classification ?Where one image can contains multiple class.

Adrian Rosebrock
September 5, 2019 at 9:58 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-547062)

I already covered **multi-label image classification. (https://www.pyimagesearch.com/2018/05/07/multi-label-classification-with-keras/)**

Adarsh (http://aidiotlabs.ai)
September 7, 2019 at 3:53 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-547938)

Is there a crude way to estimate how many sample images we need for training for a given number of classes. Or, is it that we should keep trying out until we edge on required accuracy.

Adrian Rosebrock

September 12, 2019 at 11:14 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-550292)

Take a look at **Deep Learning for Computer Vision with Python (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)** where I provide my tips, suggestions, and best practices on required number of images per class.

kaki

October 2, 2019 at 2:55 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-556883)

Hi Adrian, this was a lot of help! I've got one question:

If we use method #2 the in-place augmentation in keras, does this mean our validation split of 0.25 in the train_test_split call will remain true even after augmenting the training data?

Adrian Rosebrock

October 3, 2019 at 12:18 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-557429)

I'm not sure I fully understand your question. You don't apply data augmentation to your validation set. And furthermore, your validation set should *always* be separate from your training set. You should never mix the two.

Kaki

October 3, 2019 at 8:23 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-557675)

Sorry let me clarify. Does they 0.25 split ratio still stand even after we've split the training and validation sets AND then augmented the training data using method #2?

Pand

October 4, 2019 at 9:14 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-557905)

Hey Adrian! One question: I have a model which inputs an image and outputs 5 arrays (heatmaps). Is there a way to adapt the outputs accordingly with the transformation each input image received? Thanks and great work!!

Eduard Bulava

October 14, 2019 at 9:11 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-564162)

Hello everyone, I trained the model based on Adrian's tutorials. Unfortunately, now my model predicts the same result despite the incoming data. Spent a lot of time trying to figure out what I did wrong. Could someone point out my mistake? I will be very grateful.

Akash Desarda

March 24, 2020 at 8:39 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-767462)

Instead of reading all image using OpenCV then iterating to create labels and data, why not use 'flow_from_directory()' instead of 'flow()' method of ImageDataGenerator API.

Adrian Rosebrock

March 25, 2020 at 1:35 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-767647)

You can, but if your dataset can fit into memory you should load it into RAM as that reduces I/O operations and makes training faster.

Yonten Jamtsho

April 9, 2020 at 3:40 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-772133)

While splitting the datasets, does 75-25% ratio applied to Cat and Dog images evenly?

Adrian Rosebrock

April 9, 2020 at 8:55 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-772448)

If you stratify the labels during the split, yes.

Jesus Claus

April 9, 2020 at 8:21 pm (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-773353)

Thanks for explaining this. Every tutorial I've seen doesn't give any further explanation than "ImageDataGenerator augments data." Like wow, f**king thanks for that. Might as well say X does Y to Z.

Adrian Rosebrock

April 16, 2020 at 8:09 am (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/#comment-785505)

Thanks Jesus, I'm so happy you found the tutorial helpful!

Comment section

Hey, Adrian Rosebrock here, author and creator of PyImageSearch. While I love hearing from readers, a couple years ago I made the tough decision to no longer offer 1:1 help over blog post comments.

At the time I was receiving 200+ emails per day and another 100+ blog post comments. I simply did not have the time to moderate and respond to them all, and the sheer volume of requests was taking a toll on me.

Instead, my goal is to *do the most good* for the computer vision, deep learning, and OpenCV community at large by focusing my time on authoring high-quality blog posts, tutorials, and books/courses.

If you need help learning computer vision and deep learning, I suggest you refer to my full catalog of books

If you need help learning computer vision and deep learning, I suggest you refer to my full catalog of books and courses (https://www.pyimagesearch.com/books-and-courses/) — they have helped tens of thousands of developers, students, and researchers *just like yourself* learn Computer Vision, Deep Learning, and OpenCV.

Click here to browse my full catalog. (https://www.pyimagesearch.com/books-and-courses/)

Similar articles

RESOURCES TUTORIALS

Fast, optimized 'for' pixel loops with OpenCV and Python

August 28, 2017

(https://www.pyimagesearch.com/2017/08/28/fast-optimized-for-pixel-loops-with-opencv-and-python/)

IMAGE PROCESSING TUTORIALS

OpenCV and Python Color Detection

August 4, 2014

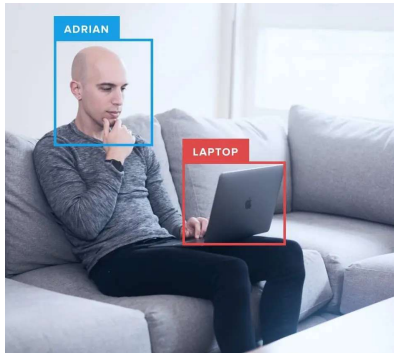
(https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/)

IMAGE PROCESSING IMAGE SEARCH ENGINE BASICS TUTORIALS

Building an Image Hashing Search Engine with VP-Trees and OpenCV

August 26, 2019

(https://www.pyimagesearch.com/2019/08/26/building-an-image-hashing-search-engine-with-vp-trees-and-opencv/)



You can learn Computer Vision, Deep Learning, and OpenCV.

Get your FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF. Inside you'll find our hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

Topics

Machine Learning and Computer Vision (https://www.pyimagesearch.com/category/machine-learning-2/)

Deep Learning (https://www.pyimagesearch.com/category/deep-learning-2/)

Dlib Library (https://www.pyimagesearch.com/category/dlib/)

Embedded/IoT and Computer Vision (https://www.pyimagesearch.com/category/embedded/)

Face Applications (https://www.pyimagesearch.com/category/faces/)

Image Processing (https://www.pyimagesearch.com/category/image-processing/)

Interviews (https://www.pyimagesearch.com/category/interviews/)

Keras (https://www.pyimagesearch.com/category/keras/)

Medical Computer Vision (https://www.pyimagesearch.com/category/medical/)

Optical Character Recognition (OCR) (https://www.pyimagesearch.com/category/optical-character-recognition-ocr/)

Object Detection (https://www.pyimagesearch.com/category/object-detection/)

Object Tracking (https://www.pyimagesearch.com/category/object-tracking/)

OpenCV Tutorials (https://www.pyimagesearch.com/category/opencv/)

Raspberry Pi (https://www.pyimagesearch.com/category/raspberry-pi/)

Books & Courses

FREE CV, DL, and OpenCV Crash Course (https://www.pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/)

Practical Python and OpenCV (https://www.pyimagesearch.com/practical-python-opencv/)

Deep Learning for Computer Vision with Python

PyImageSearch

Get Started (https://www.pyimagesearch.com/start-here/)

OpenCV Install Guides (https://www.pyimagesearch.com/opencv-tutorials-resources-guides/)

About (https://www.pyimagesearch.com/about/)

FAQ (https://www.pyimagesearch.com/faqs/)

Deep Learning for Computer Vision with Python (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)

PyImageSearch Gurus Course (https://www.pyimagesearch.com/pyimagesearch-gurus/)

Raspberry Pi for Computer Vision (https://www.pyimagesearch.com/raspberry-pi-for-computer-vision/)

Blog (https://www.pyimagesearch.com/topics/)

Contact (https://www.pyimagesearch.com/contact/)

Privacy Policy (https://www.pyimagesearch.com/privacy-policy/)

(https://www.facebook.com/pyimagesearch) (https://twitter.com/PyImageSearch) (http://www.linkedin.com/pub/adrian-rosebrock/2a/873/59b) (https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMCk_Q/videos)