



search

Home

+=1

Support the Content

Community

Log in

Sign up

Loading in your own data - Deep Learning basics with Python, TensorFlow and Keras p.2

Loading in your own data - Deep Learning b...



Welcome to a tutorial where we'll be discussing how to load in our own outside datasets, which comes with all sorts of challenges!

First, we need a dataset. Let's grab the [Dogs vs Cats dataset from Microsoft](#). If this dataset disappears, someone let me know. I will host it myself.

Now that you have the dataset, it's currently compressed. Unzip the dataset, and you should find that it creates a directory called `PetImages`. Inside of that, we have `Cat` and `Dog` directories, which are then filled with images of cats and dogs. Easy enough! Let's play with this dataset! First, we need to understand how we will convert this dataset to training data. We have a few issues right out of the gate. The largest issue is not all of these images are the same size. While we can eventually have variable-sized layers in neural networks, this is not the most basic thing to achieve. We're going to want to reshape things for now so every image has the same dimensions. Next, we may or may not want to keep color. To begin, install `matplotlib` if you don't already have it (`pip install matplotlib`), as well as `opencv` (`pip install opencv-python`).



search

Home

+1

Support the Content

Community

Log in

Sign up

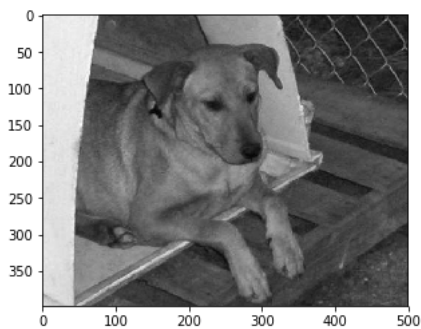
```
import cv2
from tqdm import tqdm

DATADIR = "X:/Datasets/PetImages"

CATEGORIES = ["Dog", "Cat"]

for category in CATEGORIES: # do dogs and cats
    path = os.path.join(DATADIR,category) # create path to dogs and cats
    for img in os.listdir(path): # iterate over each image per dogs and cats
        img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE) # convert to array
        plt.imshow(img_array, cmap='gray') # graph it
        plt.show() # display!

        break # we just want one for now so break
    break #...and one more!
```



Oh look, a dog!

```
print(img_array)

[[189 189 189 ... 29 29 31]
 [186 186 186 ... 36 35 36]
 [184 185 185 ... 35 33 33]
 ...
 [168 169 170 ... 71 72 72]
 [169 170 171 ... 68 67 67]
 [168 169 170 ... 64 63 62]]
```

And now it's shape:

```
print(img_array.shape)

(398, 500)
```

So that's a 375 tall, 500 wide, and 3-channel image. 3-channel is because it's RGB (color). We definitely don't want the images that big, but also various images are different shapes, and this is also a problem.



search

Home

+=1

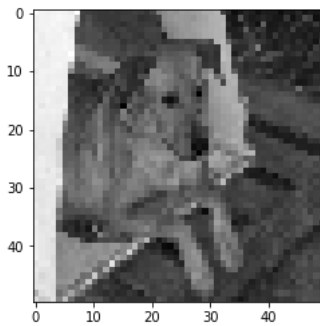
Support the Content

Community

Log in

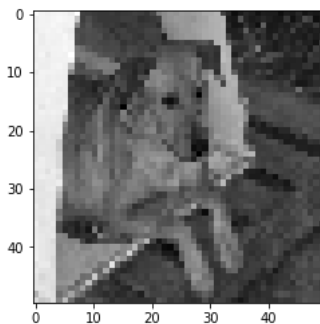
Sign up

```
plt.imshow(new_array, cmap='gray')  
plt.show()
```



Hmm, that's a bit blurry I'd say. Let's go with 100x100?

```
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  
plt.imshow(new_array, cmap='gray')  
plt.show()
```



Better. Let's try that. Next, we're going to want to create training data and all that, but, first, we should set aside some images for final testing. I am going to just manually create a directory called `Testing` and then create 2 directories inside of there, one for `Dog` and one for `Cat`. From here, I am just going to move the first 15 images from both `Dog` and `Cat` into the training versions. Make sure you move them, not copy. We will use this for our final tests.

Now, we want to begin building our training data!



search

Home

+1

Support the Content

Community

Log in

Sign up

```
for category in CATEGORIES: # do dogs and cats
```

```
    path = os.path.join(DATADIR,category) # create path to dogs and cats
```

```
    class_num = CATEGORIES.index(category) # get the classification (0 or a 1). 0=dog 1=cat
```

```
    for img in tqdm(os.listdir(path)): # iterate over each image per dogs and cats
```

```
        try:
```

```
            img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE) # convert to array
```

```
            new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE)) # resize to normalize data size
```

```
            training_data.append([new_array, class_num]) # add this to our training_data
```

```
        except Exception as e: # in the interest in keeping the output clean...
```

```
            pass
```

```
        #except OSError as e:
```

```
        #     print("OSErrorBad img most likely", e, os.path.join(path,img))
```

```
        #except Exception as e:
```

```
        #     print("general exception", e, os.path.join(path,img))
```

```
create_training_data()
```

```
print(len(training_data))
```

```
100%|a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a| 12486/12486 [00:14<00:00, 890.92it/s]
```

```
100%|a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a| 12486/12486 [00:12<00:00, 966.98it/s]
```

```
24916
```

Great, we have almost 25K samples! That's awesome.

One thing we want to do is make sure our data is balanced. In the case of this dataset, I can see that the dataset started off as being balanced. By balanced, I mean there are the same number of examples for each class (same number of dogs and cats). If not balanced, you either want to pass the class weights to the model, so that it can measure error appropriately, or balance your samples by trimming the larger set to be the same size as the smaller set.

If you do not balance, the model will initially learn that the best thing to do is predict only one class, whichever is the most common. Then, it will often get stuck here. In our case though, this data is already balanced, so that's easy enough. Maybe later we'll have a dataset that isn't balanced so nicely.

Also, if you have a dataset that is too large to fit into your ram, you can batch-load in your data. There are many ways to do this, some outside of TensorFlow and some built in. We may discuss this further, but, for now, we're mainly trying to cover how your data should look, be shaped, and fed into the models.

Next, we want to shuffle the data. Right now our data is just all dogs, then all cats. This will usually wind up causing trouble too, as, initially, the classifier will learn to just predict dogs always. Then it will shift to oh, just predict all cats! Going back and forth like this is no good either.

```
import random
```

```
random.shuffle(training_data)
```

Our training_data is a list, meaning it's mutable, so it's now nicely shuffled. We can confirm this by iterating over a few of the initial samples and printing out the class.

```
for sample in training_data[:10]:
```

```
    print(sample[1])
```

```
1
1
1
0
0
1
0
0
0
1
```



search

Home

+=1

Support the Content

Community

Log in

Sign up

y = []

```
for features,label in training_data:
    X.append(features)
    y.append(label)
```

```
print(X[0].reshape(-1, IMG_SIZE, IMG_SIZE, 1))
```

```
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

```
[[[215]
  [113]
  [169]
  ...
  [100]
  [ 62]
  [ 83]]

 [128]
 [140]
 [156]
  ...
 [113]
 [ 63]
 [ 97]]

 [149]
 [146]
 [183]
  ...
 [141]
 [ 90]
 [ 98]]

 ...

 [[ 58]
  [ 20]
  [ 23]
  ...
  [ 79]
 [208]
 [209]]

 [[ 35]
  [ 26]
  [ 21]
  ...
  [ 77]
 [211]
 [210]]

 [[ 27]
  [ 21]
  [ 24]
  ...
  [ 93]
 [205]
 [195]]]]
```

Let's save this data, so that we don't need to keep calculating it every time we want to play with the neural network model:



search

[Home](#)[+=1](#)[Support the Content](#)[Community](#)[Log in](#)[Sign up](#)

```
pickle.dump(X, pickle_out)
pickle_out.close()
```

```
pickle_out = open("y.pickle", "wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

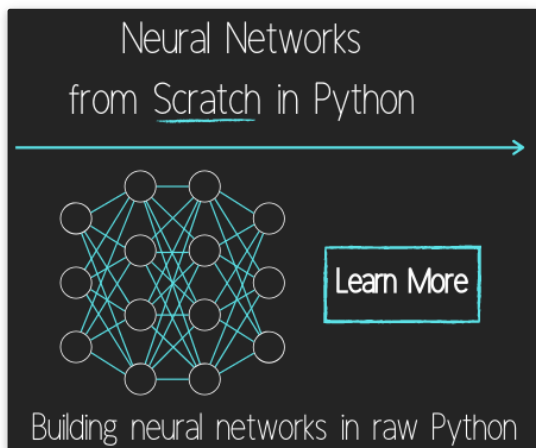
We can always load it in to our current script, or a totally new one by doing:

```
pickle_in = open("X.pickle", "rb")
X = pickle.load(pickle_in)
```

```
pickle_in = open("y.pickle", "rb")
y = pickle.load(pickle_in)
```

Now that we've got our dataset, we're ready to cover convolutional neural networks and implement one with our data for classification.

The next tutorial: [Convolutional Neural Networks - Deep Learning Basics With Python, TensorFlow And Keras P.3](#)



[Introduction to Deep Learning - Deep Learning basics with Python, TensorFlow and Keras p.1](#)

[Loading in your own data - Deep Learning basics with Python, TensorFlow and Keras p.2](#)

[Convolutional Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.3](#)

[Analyzing Models with TensorBoard - Deep Learning basics with Python, TensorFlow and Keras p.4](#)


[Optimizing Models with TensorBoard - Deep Learning basics with Python, TensorFlow and Keras p.5](#)


[How to use your trained model - Deep Learning basics with Python, TensorFlow and Keras p.6](#)

[Recurrent Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.7](#)

[Creating a Cryptocurrency-predicting finance recurrent neural network - Deep Learning basics with Python, TensorFlow and Keras p.8](#)

[Normalizing and creating sequences for our cryptocurrency predicting RNN - Deep Learning basics with Python, TensorFlow and Keras p.9](#)



search

Home +=1 Support the Content Community Log in Sign up

Keras p.10

Cryptocurrency-predicting RNN Model - Deep Learning basics with Python, TensorFlow and Keras p.11

An Enterprise Writing Coach

Make Your Entire Team More Effective With the Leading Writing Try For Free.

Grammarly Business

G

You've reached the end!

Contact: Harrison@pythonprogramming.net.

Support this Website!
Consulting and Contracting
[Facebook](#)
[Twitter](#)
[Instagram](#)

Legal stuff:
[Terms and Conditions](#)
[Privacy Policy](#)