Generators and Data Augmentation

February 26, 2020

1 Import packages

Note `keras.preprocessing.image`

[0]:

```
import numpy as np
from keras.preprocessing import image
import matplotlib.pyplot as plt
import imageio
```

2 Generators

The following examples are adapted from “Fluent Python” by Luciano Ramalho (2015).

Formal definition:

- A generator is a function that uses the key word `yield`

First example (unrealistic):

[68]:

```
def gen_123():
  yield '1'
  yield '2'
  yield '3'

g = gen_123()

print("type of gen_123 is " + str(type(gen_123)))
print("type of g is " + str(type(g)))
print(next(g))
print(next(g))
print(next(g))
```

```
type of gen_123 is <class 'function'>
type of g is <class 'generator'>
1
2
3
```

[69]:

```
print(next(g))
```

```
    ---------------------------------------------------------------------------

    StopIteration                             Traceback (most recent call last)

    <ipython-input-69-1dfb29d6357e> in <module>()
----> 1 print(next(g))


    StopIteration:
```

What just happened:

- gen_123 is a function
- When called, gen_123 creates a "generator object" (g)
- Calling `next(g)` basically does what you'd think calling the original gen_123 function would do.
- `yield` is similar to return: as soon as the yield statement is called, the function stops running and returns the specified quantity
- But when the generator is called again, it picks up where it left off.
- If it runs out of stuff to yield, an error is thrown.

Second Example (more like how we'll use generators):

[70]:

```
def gen_ints():
  i = 0
  while True: # generator will never stop generating another integer!
    i = i + 1
    yield i

g = gen_ints()

print(str(next(g)))
print(str(next(g)))
```

```
1
2
```

[72]:

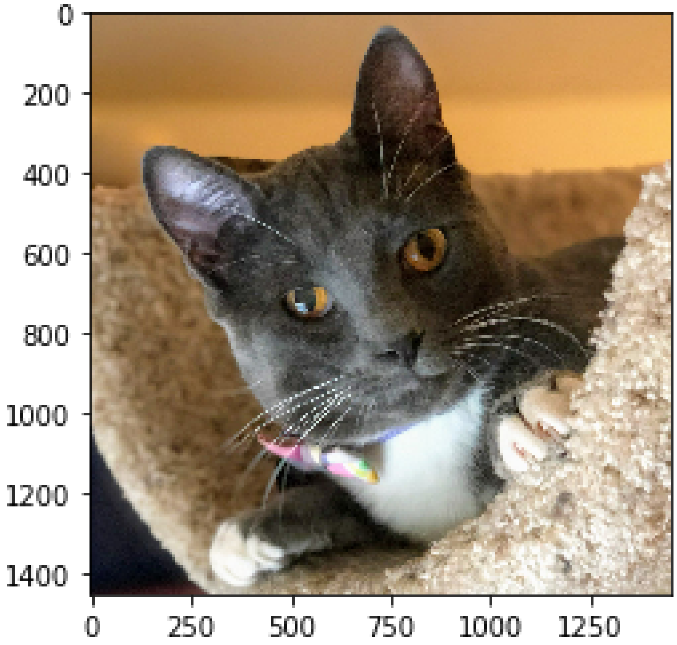
```
for i in range(10):
  print(str(next(g)))
```

```
3
4
5
6
7
8
9
10
11
12
```

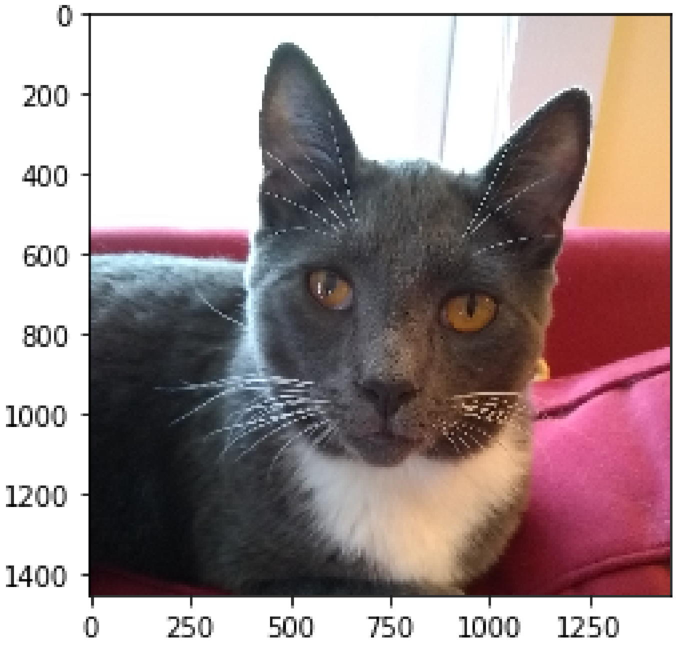
2.1 Benedict!

```
[0]: benedict_path = "http://www.evanlray.com/stat344ne_s2020/homework/hw1/benedict.
      ↪jpg"
     benedict_path2 = "https://raw.githubusercontent.com/mhc-stat344ne-s2020/
      ↪20200226_examples/master/benedict.jpg"
     benedict = np.concatenate(
         (np.array(imageio.imread(benedict_path))[np.newaxis, 50:1500, 0:1450, :],
          np.array(imageio.imread(benedict_path2))[np.newaxis, 50:1500, 0:1450, :]),
         axis = 0)
     print(benedict.shape)
     plt.imshow(benedict[0, ...])
     plt.show()
     plt.imshow(benedict[1, ...])
```

```
(2, 1450, 1450, 3)
```



```
[0]: <matplotlib.image.AxesImage at 0x7f8598aa6da0>
```



`benedict` is in the shape expected by Keras (observations in first axis):

- 2 observations
- 1450 rows, 1450 columns, 3 channels (RGB)

2.2 Create Image Data Generator

- It's a generator.
- Every time you call it, you get a new (batch of) images.
- It never runs out (it goes back to the beginning of the data set, shuffles the data, and starts over).

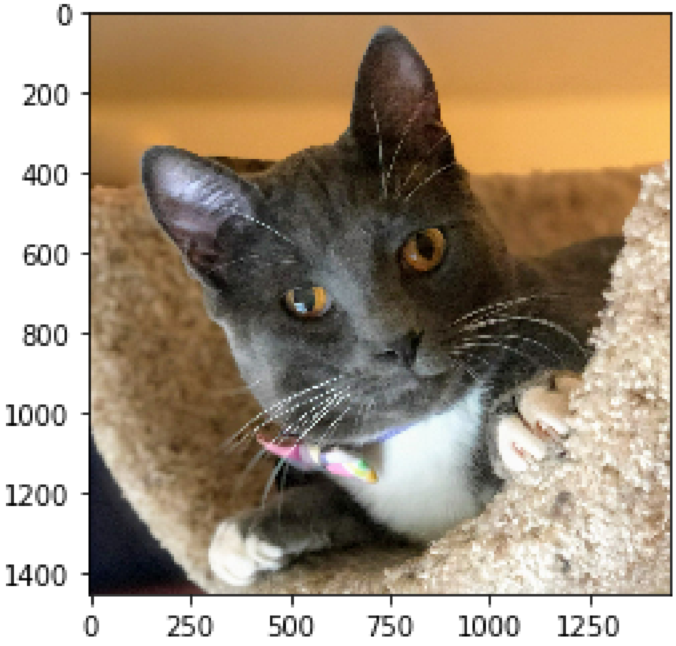
[0]:

```
# create data generator
train_X = benedict
train_y = np.array([1, 1])
data_gen_setup = image.ImageDataGenerator()
gen = data_gen_setup.flow(train_X, train_y, batch_size=1)
```

[74]:

```
batch = next(gen)
X = batch[0].astype(int)
y = batch[1]
print(X.shape)
plt.imshow(X[0, :, :, :])
print(y)
```

```
(1, 1450, 1450, 3)
[1]
```

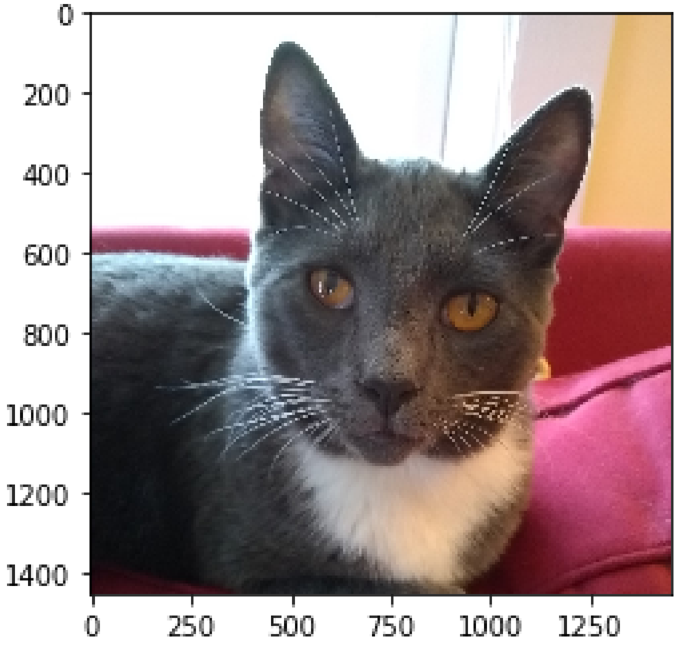


[75]:

```
batch = next(gen)
X = batch[0].astype(int)
y = batch[1]
```

```
print(X.shape)
plt.imshow(X[0, :, :, :])
print(y)
```

```
(1, 1450, 1450, 3)
[1]
```



We could have set it up to ask for batches of size 2:

```
[76]: gen2 = data_gen_setup.flow(train_X, train_y, batch_size=2)
      batch = next(gen2)
      X = batch[0].astype(int)
      print(X.shape)
```

```
(2, 1450, 1450, 3)
```

You could also "flow" the files from a directory containing the image files. You'll see this in today's lab.

3 Data Augmentation

- You're working with image data (occasionally applied in other settings, but relatively rare)
- Your data set is small and/or you're overfitting
- Take your original training set images (not validation or test set), and randomly modify them a little.

Examples below adapted from https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

3.1 Rotation

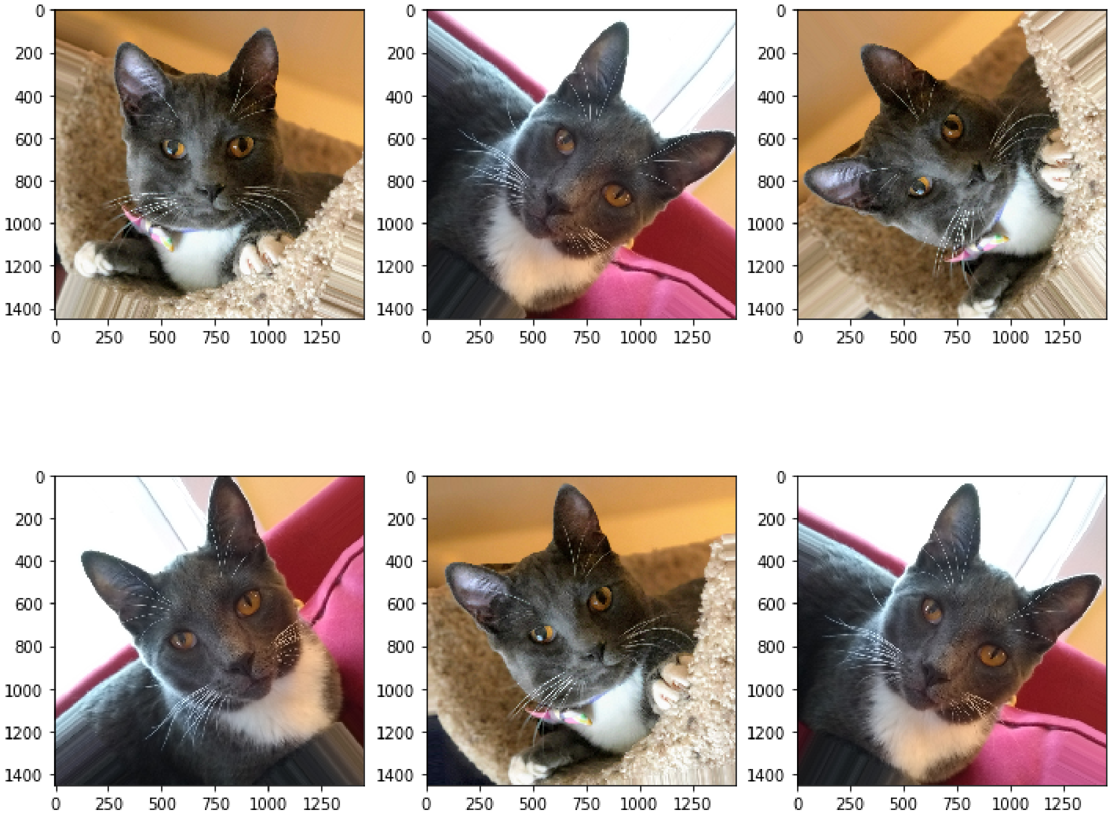
[77]:

```
data_gen_setup = image.ImageDataGenerator(
      rotation_range=40,
      fill_mode='nearest')

train_generator = data_gen_setup.flow(train_X, train_y, batch_size=1)

fig, axs = plt.subplots(2, 3, figsize = (12, 10))

for i in range(2):
  for j in range(3):
    batch = next(train_generator)
    X = batch[0].astype(int)
    axs[i, j].imshow(X[0, :, :, :])
```



3.1.1 Width Shift

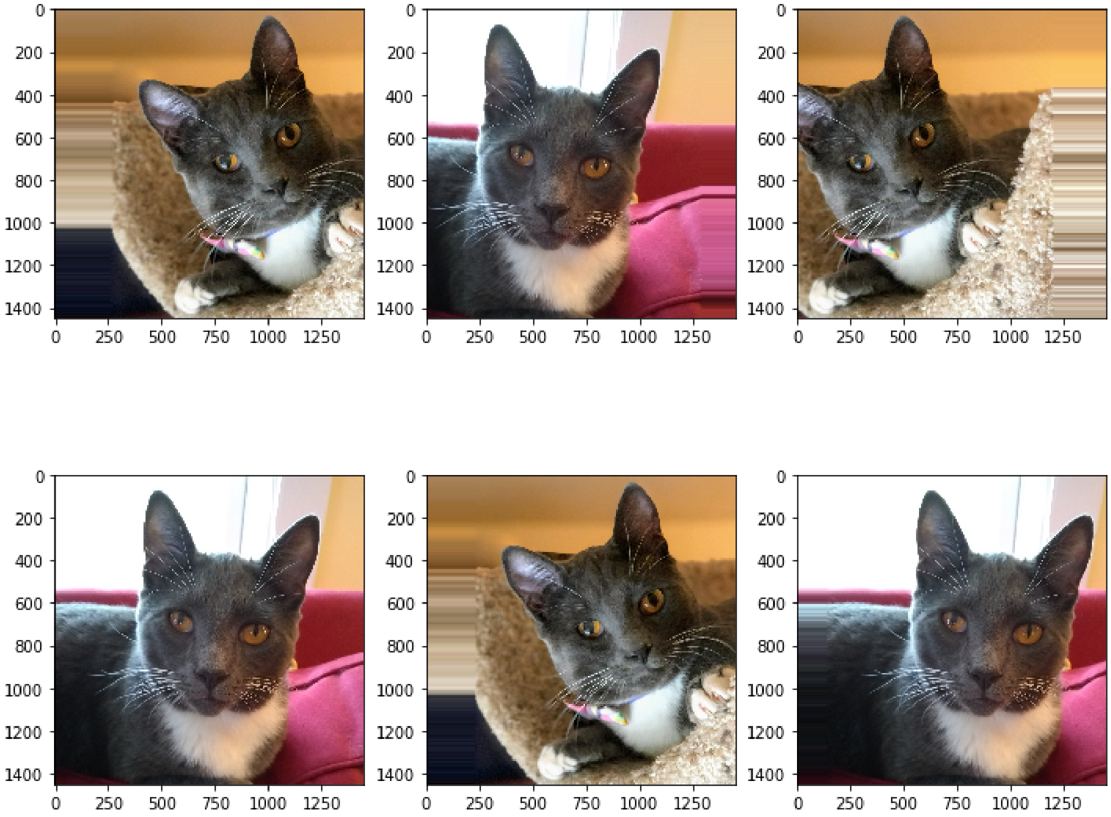
[79]:

```
data_gen_setup = image.ImageDataGenerator(
    width_shift_range=0.2,
    fill_mode='nearest')

train_generator = data_gen_setup.flow(train_X, train_y, batch_size=1)

fig, axs = plt.subplots(2, 3, figsize = (12, 10))

for i in range(2):
  for j in range(3):
    batch = next(train_generator)
    X = batch[0].astype(int)
    axs[i, j].imshow(X[0, :, :, :])
```



3.1.2 Height Shift

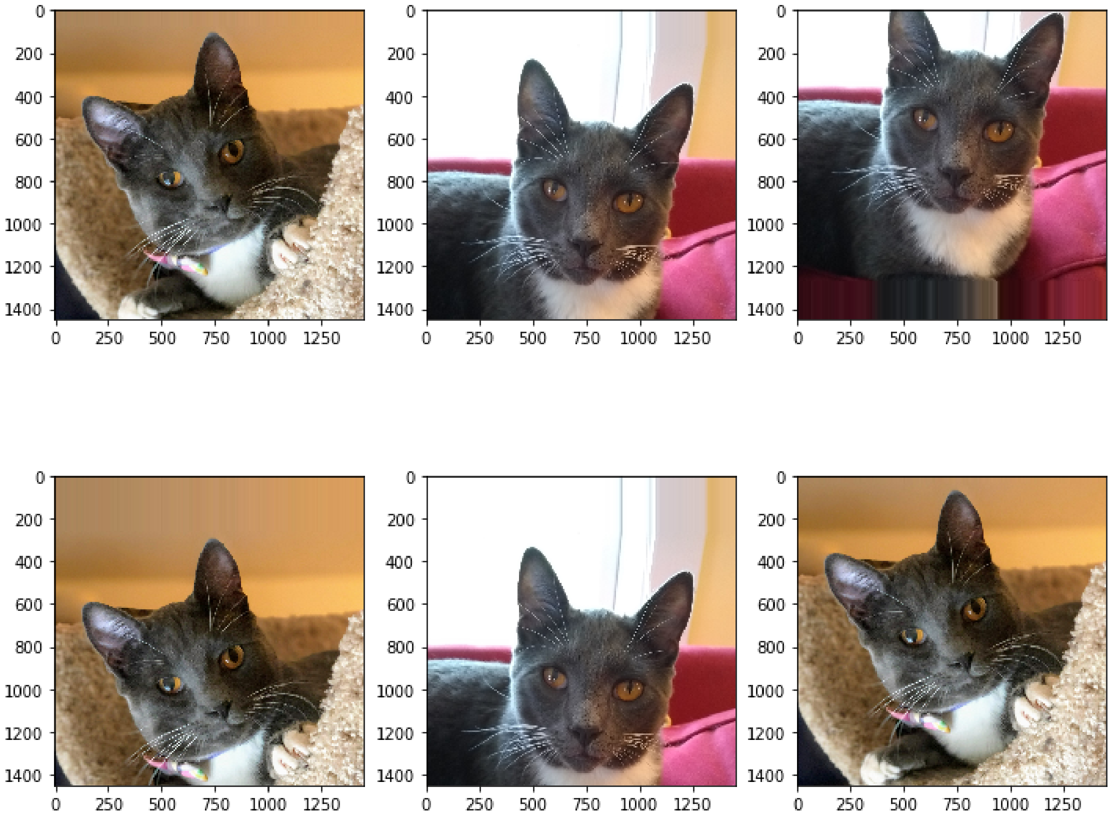
[81]:

```
data_gen_setup = image.ImageDataGenerator(
    height_shift_range=0.2,
    fill_mode='nearest')

train_generator = data_gen_setup.flow(train_X, train_y, batch_size=1)

fig, axs = plt.subplots(2, 3, figsize = (12, 10))

for i in range(2):
  for j in range(3):
    batch = next(train_generator)
    X = batch[0].astype(int)
    axs[i, j].imshow(X[0, :, :, :])
```



3.1.3 Shear

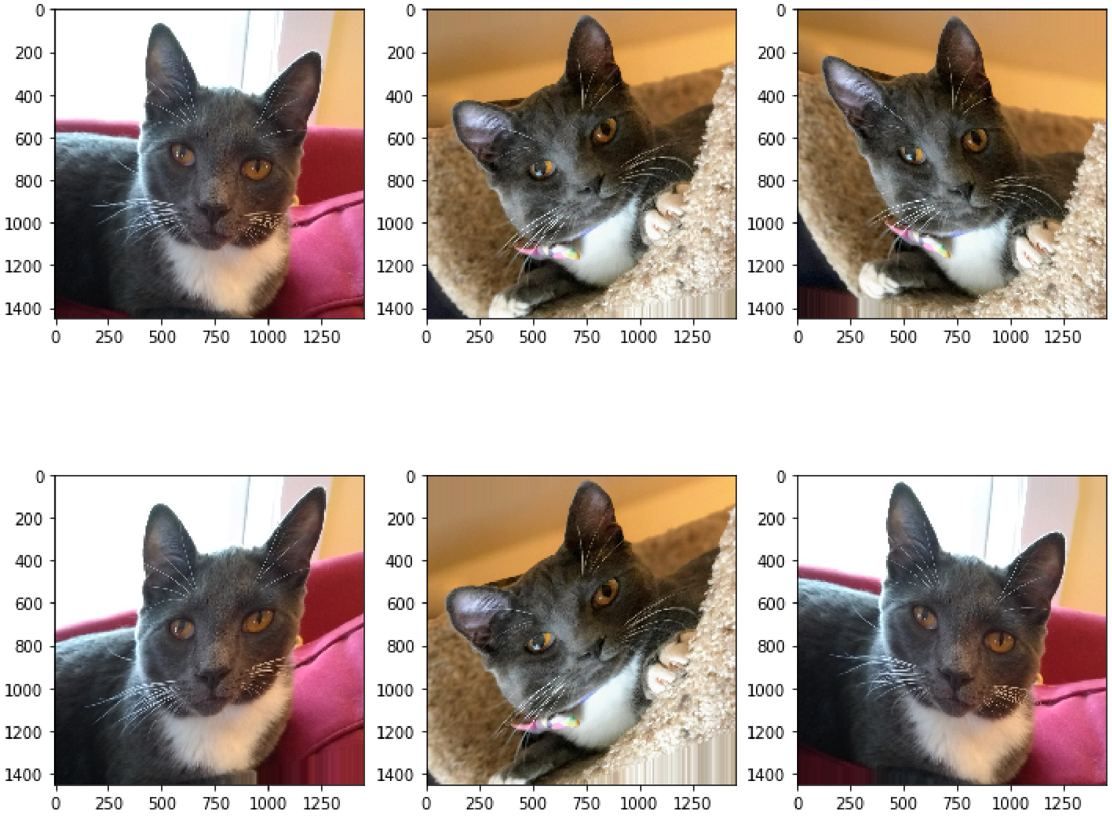
[82]:

```
data_gen_setup = image.ImageDataGenerator(
    shear_range=20,
    fill_mode='nearest')

train_generator = data_gen_setup.flow(train_X, train_y, batch_size=1)

fig, axs = plt.subplots(2, 3, figsize = (12, 10))

for i in range(2):
  for j in range(3):
    batch = next(train_generator)
    X = batch[0].astype(int)
    axs[i, j].imshow(X[0, :, :, :])
```



3.1.4 Horizontal Flip

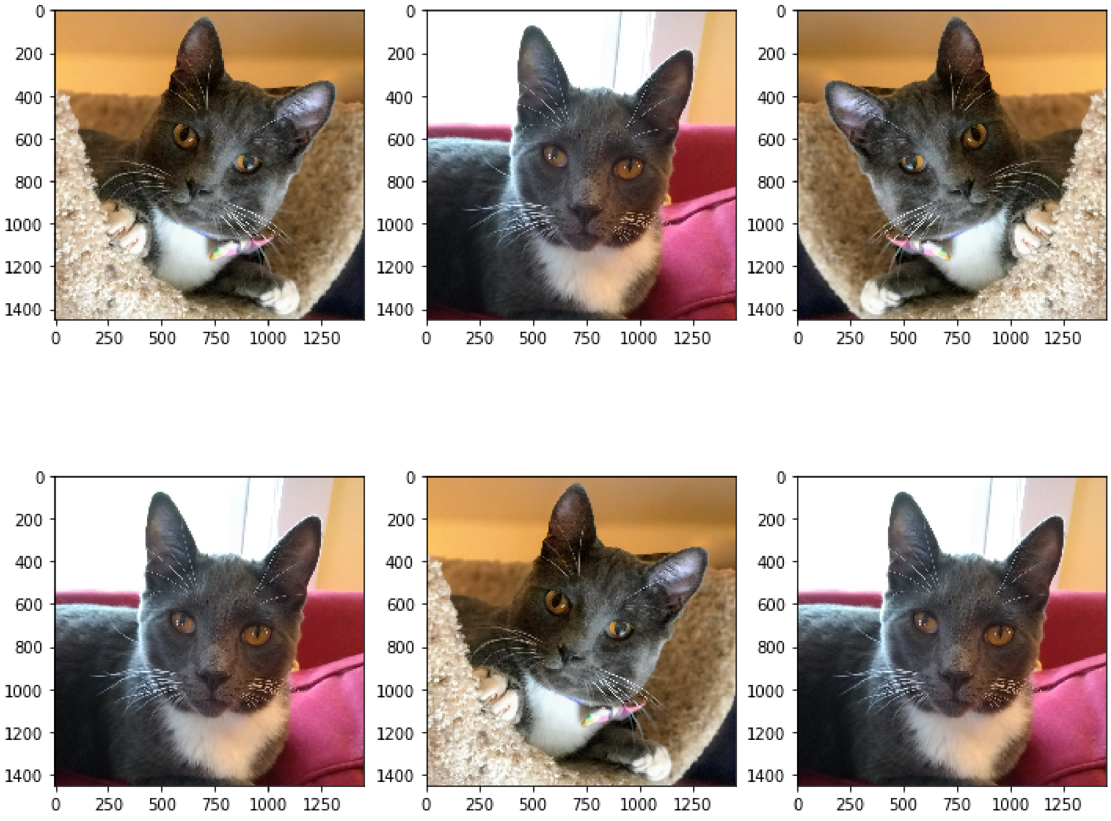
[83]:

```
data_gen_setup = image.ImageDataGenerator(
    horizontal_flip=True,
    fill_mode='nearest')

train_generator = data_gen_setup.flow(train_X, train_y, batch_size=1)

fig, axs = plt.subplots(2, 3, figsize = (12, 10))

for i in range(2):
  for j in range(3):
    batch = next(train_generator)
    X = batch[0].astype(int)
    axs[i, j].imshow(X[0, :, :, :])
```



3.1.5 All at Once

[84]:

```python
data_gen_setup = image.ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=20,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

train_generator = data_gen_setup.flow(train_X, train_y, batch_size=1)

fig, axs = plt.subplots(2, 3, figsize = (12, 10))

for i in range(2):
  for j in range(3):
    batch = next(train_generator)
    X = batch[0].astype(int)
    axs[i, j].imshow(X[0, :, :, :])
```