Working with Custom Images

So far everything we've worked with has been nicely formatted for us already by Keras.

Let's explore what its like to work with a more realistic data set.

The Data

PLEASE NOTE: THIS DATASET IS VERY LARGE. IT CAN BE DOWNLOADED FROM THE PREVIOUS LECTURE. PLEASE WATCH THE VIDEO LECTURE ON HOW TO GET THE DATA.

USE OUR VERSION OF THE DATA. WE ALREADY ORGANIZED IT FOR YOU!!

ORIGINAL DATA SOURCE:

https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765 (https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765)

The Kaggle Competition: Cats and Dogs (https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition) includes 25,000 images of cats and dogs. We will be building a classifier that works with these images and attempt to detect dogs versus cats!

The pictures are numbered 0-12499 for both cats and dogs, thus we have 12,500 images of Dogs and 12,500 images of Cats. This is a huge dataset!!

**Note: We will be dealing with real image files, NOT numpy arrays. Which means a large part of this process will be learning how to work with and deal with large groups of image files. This is too much data to fit in memory as a numpy array, so we'll need to feed it into our model in batches. **

Visualizing the Data

Let's take a closer look at the data.

In [1]:
```
import matplotlib.pyplot as plt
import cv2
# Technically not necessary in newest versions of jupyter
%matplotlib inline
```

In [2]:
```
cat4 = cv2.imread(r'C:\Users\ACER\Downloads\CATS_DOGS\train\CAT\4.jpg')
cat4 = cv2.cvtColor(cat4, cv2.COLOR_BGR2RGB)
```

In [3]:
```
type(cat4)
```

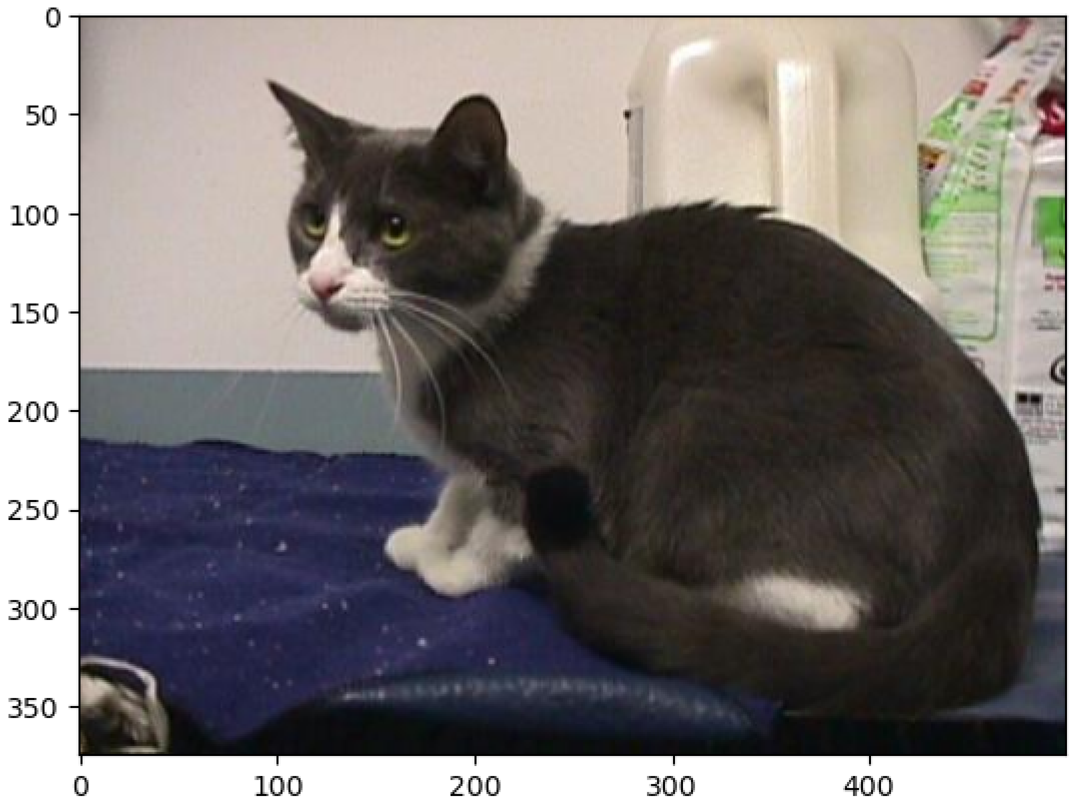
Out[3]:
```
numpy.ndarray
```

In [4]:
```
cat4.shape
```

Out[4]:
```
(375, 500, 3)
```

In [5]:
```
plt.imshow(cat4)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x1daa1dd83d0>
```



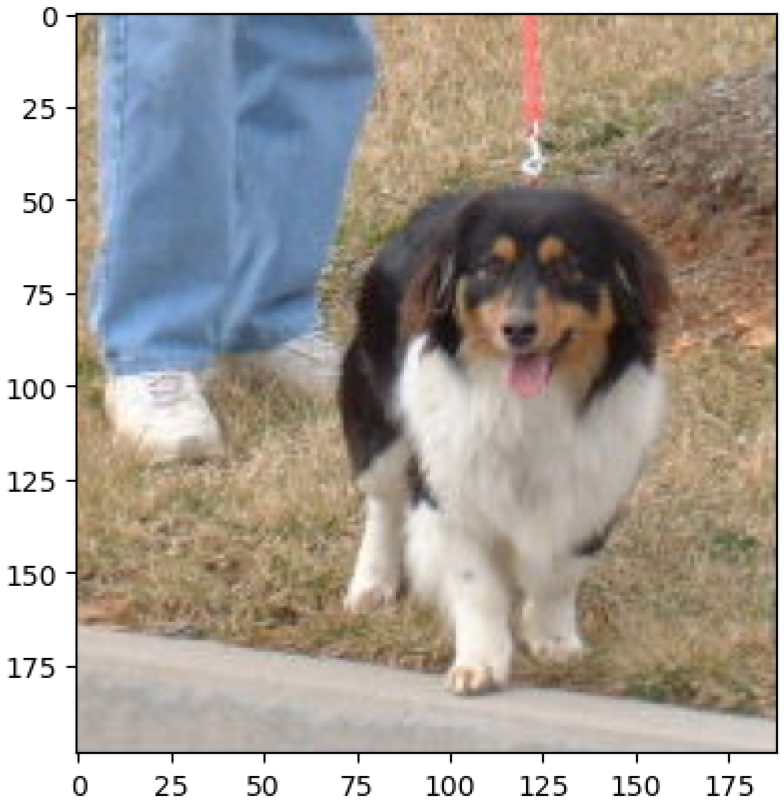
In [6]:
```
dog2 = cv2.imread(r'C:\Users\ACER\Downloads\CATS_DOGS\train\DOG\2.jpg')
dog2 = cv2.cvtColor(dog2, cv2.COLOR_BGR2RGB)
```

In [7]:
```
dog2.shape
```

Out[7]:
```
(199, 188, 3)
```

In [8]:
```
plt.imshow(dog2)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x1daa26e9f60>
```



Preparing the Data for the model

There is too much data for us to read all at once in memory. We can use some built in functions in Keras to automatically process the data, generate a flow of batches from a directory, and also manipulate the images.

Image Manipulation

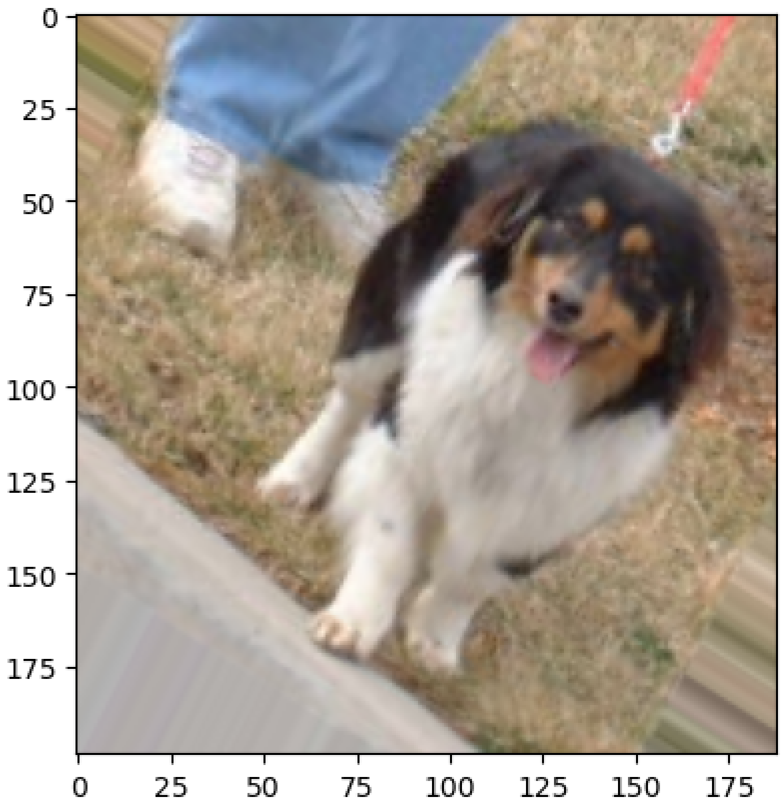
Its usually a good idea to manipulate the images with rotation, resizing, and scaling so the model becomes more robust to different images that our data set doesn't have. We can use the **ImageDataGenerator** to do this automatically for us. Check out the documentation for a full list of all the parameters you can use here!

In [9]:
```
from keras.preprocessing.image import ImageDataGenerator
```

In [10]:
```
image_gen = ImageDataGenerator(rotation_range = 30,
                              width_shift_range = 0.1,
                              height_shift_range = 0.1,
                              rescale = 1/255,
                              shear_range = 0.2,
                              zoom_range = 0.2,
                              horizontal_flip = True,
                              fill_mode = 'nearest')
```

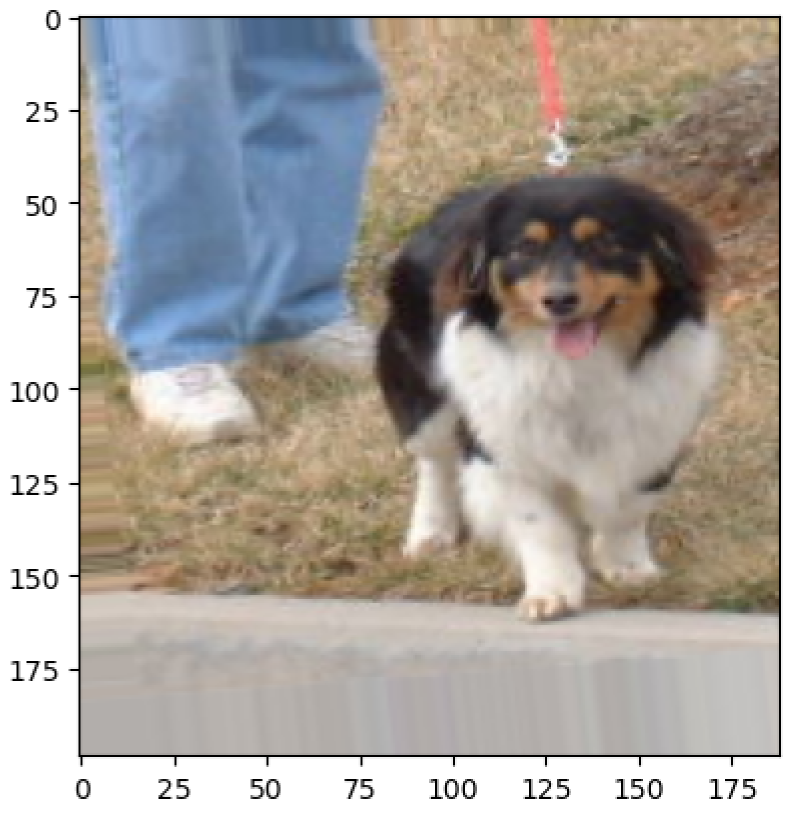
In [11]:
```
plt.imshow(image_gen.random_transform(dog2))
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x1daacbb83a0>
```



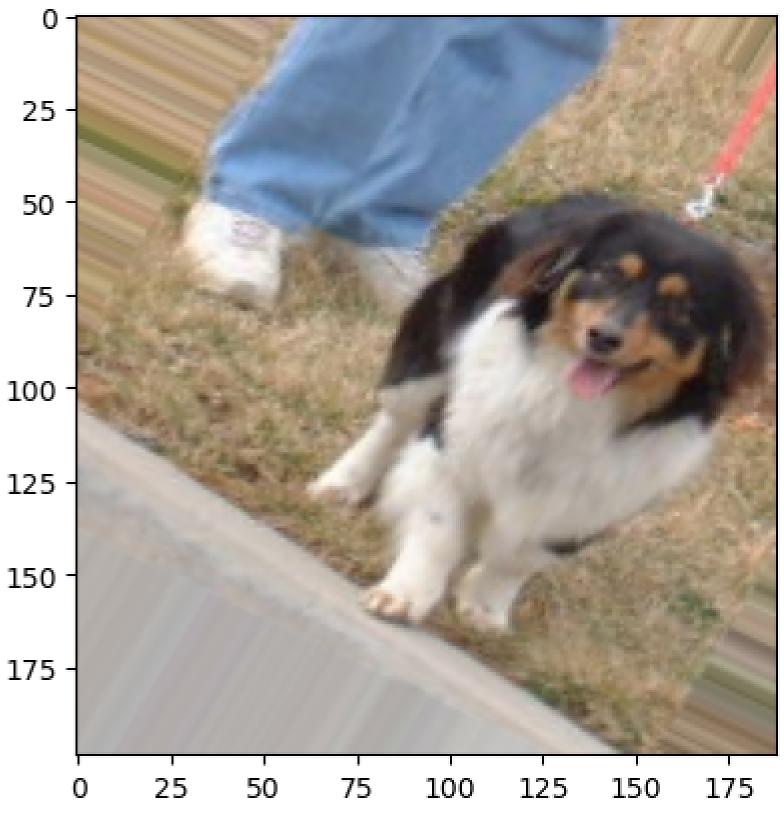
In [12]:
```
plt.imshow(image_gen.random_transform(dog2))
```

Out[12]:
```
<matplotlib.image.AxesImage at 0x1daacc9bac0>
```



In [13]:
```
plt.imshow(image_gen.random_transform(dog2))
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x1daade333d0>
```



Generating many manipulated images from a directory

In order to use .flow_from_directory, you must organize the images in sub-directories. This is an absolute requirement, otherwise the method won't work. The directories should only contain images of one class, so one folder per class of images.

Structure Needed:

- Image Data Folder
    - Class 1
        - 0.jpg
        - 1.jpg
        - ...
    - Class 2
        - 0.jpg
        - 1.jpg
        - ...
    - ...
    - Class n

In [14]:
```
image_gen.flow_from_directory(r'C:\Users\ACER\Downloads\CATS_DOGS\train')
```

```
Found 18743 images belonging to 2 classes.
```

Out[14]:
```
<keras.src.preprocessing.image.DirectoryIterator at 0x1daade57f40>
```

In [15]:
```
image_gen.flow_from_directory(r'C:\Users\ACER\Downloads\CATS_DOGS\test')
```

```
Found 6251 images belonging to 2 classes.
```

Out[15]:
```
<keras.src.preprocessing.image.DirectoryIterator at 0x1daadec25f0>
```

Resizing Images

Let's have Keras resize all the images to 150 pixels by 150 pixels once they've been manipulated.

In [16]:
```
# width,height,channels
image_shape = (150, 150, 3)
```

Creating the Model

In [17]:
```
from keras.models import Sequential
from keras.layers import Activation, Dropout, Flatten, Dense, Conv2D, MaxPooli
```

In [18]:
```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (150, 150, 
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(filters = 64, kernel_size = (3, 3), input_shape = (150, 150, 
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(filters = 64, kernel_size = (3, 3), input_shape = (150, 150, 
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten())

model.add(Dense(128))
model.add(Activation('relu'))

# Dropouts help reduce overfitting by randomly turning neurons off during trai
# Here we say randomly turn off 50% of neurons.
model.add(Dropout(0.5))

# Last layer, remember its binary, 0=cat , 1=dog
model.add(Dense(1))
model.add(Activation('softmax'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
```

In [19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        36928

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 64)        0
 g2D)

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 128)               2367616

 activation (Activation)     (None, 128)               0

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

 activation_1 (Activation)   (None, 1)                 0

=================================================================
Total params: 2424065 (9.25 MB)
Trainable params: 2424065 (9.25 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training the Model

In [20]:
```
batch_size = 16
train_image_gen = image_gen.flow_from_directory(r'C:\Users\ACER\Downloads\CATS_
                                                target_size = image_shape[:2],
                                                batch_size = batch_size,
                                                class_mode = 'binary')
```

```
Found 18743 images belonging to 2 classes.
```

In [21]:
```
# why only height and width of the image for target size?
```

In [22]:
```
batch_size = 16
test_image_gen = image_gen.flow_from_directory(r'C:\Users\ACER\Downloads\CATS_I
                                               target_size = image_shape[:2],
                                               batch_size = batch_size,
                                               class_mode = 'binary')
```

```
Found 6251 images belonging to 2 classes.
```

In [23]:
```
train_image_gen.class_indices
```

Out[23]:
```
{'CAT': 0, 'DOG': 1}
```

In [24]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [25]:
```
results = model.fit_generator(train_image_gen, epochs = 10,
                              steps_per_epoch = 15,
                              validation_data = test_image_gen,
                              validation_steps = 5)
```

```
Epoch 1/10
15/15 [==============================] - 8s 505ms/step - loss: 0.7411 - acc
uracy: 0.5292 - val_loss: 0.7171 - val_accuracy: 0.5000
Epoch 2/10
15/15 [==============================] - 7s 479ms/step - loss: 0.7057 - acc
uracy: 0.4917 - val_loss: 0.7032 - val_accuracy: 0.6000
Epoch 3/10
15/15 [==============================] - 7s 485ms/step - loss: 0.6973 - acc
uracy: 0.5000 - val_loss: 0.6933 - val_accuracy: 0.4875
Epoch 4/10
15/15 [==============================] - 7s 470ms/step - loss: 0.6924 - acc
uracy: 0.5625 - val_loss: 0.6686 - val_accuracy: 0.6625
Epoch 5/10
15/15 [==============================] - 7s 476ms/step - loss: 0.6938 - acc
uracy: 0.5250 - val_loss: 0.6953 - val_accuracy: 0.4375
Epoch 6/10
15/15 [==============================] - 7s 464ms/step - loss: 0.6894 - acc
uracy: 0.5375 - val_loss: 0.6858 - val_accuracy: 0.5375
Epoch 7/10
```

In [26]:
```
# model.save('cat_dog2.h5')
```

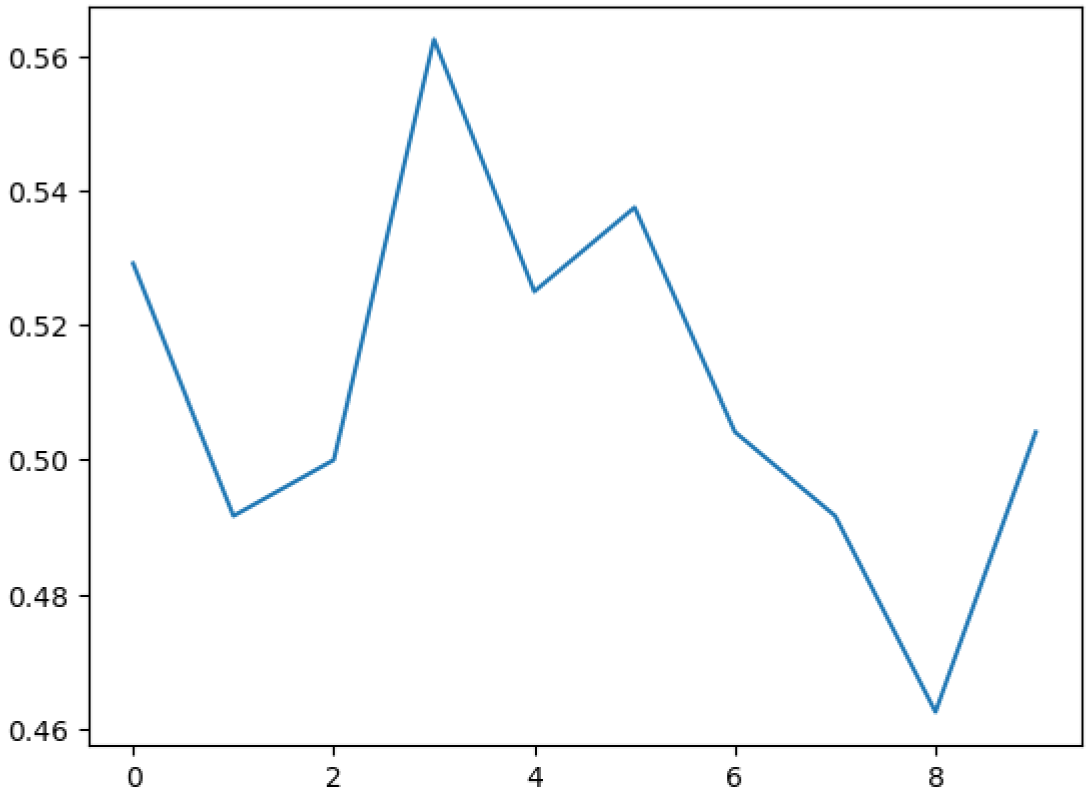
Evaluating the Model

In [27]:
```
results.history['accuracy']
```

Out[27]:
```
[0.5291666388511658,
 0.49166667461395264,
 0.5,
 0.5625,
 0.5249999761581421,
 0.5375000238418579,
 0.5041666626930237,
 0.49166667461395264,
 0.4625000059604645,
 0.5041666626930237]
```

In [28]:
```
plt.plot(results.history['accuracy'])
```

Out[28]:
```
[<matplotlib.lines.Line2D at 0x1daaf39ab00>]
```



In [29]:
```
# model.save('cat_dog_100epochs.h5')
```

Predicting on new images

In [30]:
```
train_image_gen.class_indices
```

Out[30]:
```
{'CAT': 0, 'DOG': 1}
```

In [31]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.utils import load_img, img_to_array

dog_file = r'C:\Users\ACER\Downloads\CATS_DOGS\train\DOG\2.jpg'

dog_img = tf.keras.utils.load_img(dog_file, target_size = (150, 150))
dog_img = tf.keras.utils.img_to_array(dog_img)
dog_img = np.expand_dims(dog_img, axis = 0)
dog_img = dog_img/255
```

In [32]:
```
prediction_prob = model.predict(dog_img)
```

```
1/1 [==============================] - 0s 102ms/step
```

In [33]:
```
# Output prediction
print('Probability that image is a dog is:', prediction_prob)
```

```
Probability that image is a dog is: [[1.]]
```

Great Job!