In [1]:

```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

Converting a Keras Model to JSON Format

In the previous lesson you saw how to use a CNN to make your recognition of the handwriting digits more efficient. In this lesson you'll take that to the next level, recognizing real images of Cats and Dogs in order to classify an incoming image as one or the other. In particular the handwriting recognition made your life a little easier by having all the images be the same size and shape, and they were all monochrome color. Real-world images aren't like that -- they're in different shapes, aspect ratios etc, and they're usually in color!

So, as part of the task you need to process your data -- not least resizing it to be uniform in shape.

You'll follow these steps:

1. Explore the Example Data of Cats and Dogs.
2. Build and Train a Neural Network to recognize the difference between the two.
3. Evaluate the Training and Validation accuracy.
4. Save the trained model as a Keras HDF5 file.
5. Use the tensorflow.js converter to convert the saved Keras model into JSON format.

Import Resources

In order to use the tensorflow.js converter we need to install `tensorflowjs`.

In [2]:

```
!pip install tensorflowjs
```

```
Collecting tensorflowjs
  Downloading
https://files.pythonhosted.org/packages/65/ce/8120b7d2e78c57a445b4f7ea85b034416eb3b4798c58cef13b8f6
89f/tensorflowjs-2.5.0-py3-none-any.whl (61kB)
     |████████████████████████████████| 71kB 3.4MB/s
Requirement already satisfied: h5py<3,>=2.8.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflowjs) (2.10.0)
Requirement already satisfied: tensorflow-hub<0.10,>=0.7.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflowjs) (0.9.0)
Requirement already satisfied: six<2,>=1.12.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflowjs) (1.15.0)
Requirement already satisfied: tensorflow<3,>=2.1.0 in /usr/local/lib/python3.6/dist-packages
(from tensorflowjs) (2.3.0)
Requirement already satisfied: numpy>=1.7 in /usr/local/lib/python3.6/dist-packages (from
h5py<3,>=2.8.0->tensorflowjs) (1.18.5)
Requirement already satisfied: protobuf>=3.8.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow-hub<0.10,>=0.7.0->tensorflowjs) (3.12.4)
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (1.32.0)
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (0.3.3)
Requirement already satisfied: tensorflow-estimator<2.4.0,>=2.3.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (2.3.0)
Requirement already satisfied: google-pasta>=0.1.8 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (0.2.0)
Requirement already satisfied: astunparse==1.6.3 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (1.6.3)
```

```
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (0.10.0)
Requirement already satisfied: scipy==1.4.1 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (1.4.1)
Requirement already satisfied: keras-preprocessing<1.2,>=1.1.1 in /usr/local/lib/python3.6/dist-pa
ckages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.1.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (3.3.0)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (0.35.1)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (1.1.0)
Requirement already satisfied: tensorboard<3,>=2.3.0 in /usr/local/lib/python3.6/dist-packages
(from tensorflow<3,>=2.1.0->tensorflowjs) (2.3.0)
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-packages (from
tensorflow<3,>=2.1.0->tensorflowjs) (1.12.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from
protobuf>=3.8.0->tensorflow-hub<0.10,>=0.7.0->tensorflowjs) (50.3.0)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-packages
(from tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (1.17.2)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.6/dist-p
ackages (from tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (0.4.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.6/dist-
packages (from tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (1.7.0)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from
tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (2.23.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from
tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (3.2.2)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from
tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (1.0.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages
(from google-auth<2,>=1.6.3->tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in
/usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard<3,>=2.3.0-
>tensorflow<3,>=2.1.0->tensorflowjs) (4.6)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages
(from google-auth<2,>=1.6.3->tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (4.1.1)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages
(from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs)
(1.3.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->tensorflowjs) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0-
>tensorflow<3,>=2.1.0->tensorflowjs) (1.24.3)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in
/usr/local/lib/python3.6/dist-packages (from markdown>=2.6.8->tensorboard<3,>=2.3.0-
>tensorflow<3,>=2.1.0->tensorflowjs) (2.0.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.6/dist-packages
(from pyasn1-modules>=0.2.1->google-auth<2,>=1.6.3->tensorboard<3,>=2.3.0->tensorflow<3,>=2.1.0->t
ensorflowjs) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from
requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<3,>=2.3.0-
>tensorflow<3,>=2.1.0->tensorflowjs) (3.1.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from
importlib-metadata; python_version < "3.8"->markdown>=2.6.8->tensorboard<3,>=2.3.0-
>tensorflow<3,>=2.1.0->tensorflowjs) (3.2.0)
Installing collected packages: tensorflowjs
Successfully installed tensorflowjs-2.5.0
```

In [3]:

```
!pip install tensorflow==2.0.0
```

```
Collecting tensorflow==2.0.0
  Downloading
https://files.pythonhosted.org/packages/46/0f/7bd55361168bb32796b360ad15a25de6966c9c1beb58a8e30c01c
862/tensorflow-2.0.0-cp36-cp36m-manylinux2010_x86_64.whl (86.3MB)
     |████████████████████████████████| 86.3MB 52kB/s
Collecting tensorflow-estimator<2.1.0,>=2.0.0
  Downloading
```

```
https://files.pythonhosted.org/packages/fc/08/8b927337b7019c374719145d1dceba21a8bb909b93b1ad6f8fb7d
ca1/tensorflow_estimator-2.0.1-py2.py3-none-any.whl (449kB)
     |████████████████████████████████| 450kB 51.1MB/s
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==2.0.0) (1.1.2)
Collecting gast==0.2.2
  Downloading
https://files.pythonhosted.org/packages/4e/35/11749bf99b2d4e3cceb4d55ca22590b0d7c2c62b9de38ac4a4a7f
421/gast-0.2.2.tar.gz
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (0.10.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (1.15.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (3.3.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (1.1.0)
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (1.12.1)
Collecting tensorboard<2.1.0,>=2.0.0
  Downloading
https://files.pythonhosted.org/packages/76/54/99b9d5d52d5cb732f099baaaf7740403e83fe6b0cedde940fabd2
75a/tensorboard-2.0.2-py3-none-any.whl (3.8MB)
     |████████████████████████████████| 3.8MB 47.9MB/s
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (1.18.5)
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (1.32.0)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (0.35.1)
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (0.8.1)
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (3.12.4)
Collecting keras-applications>=1.0.8
  Downloading
https://files.pythonhosted.org/packages/71/e3/19762fdfc62877ae9102edf6342d71b28fbfd9dea3d2f96a882ce
03f/Keras_Applications-1.0.8-py3-none-any.whl (50kB)
     |████████████████████████████████| 51kB 7.9MB/s
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.6/dist-packages (from
tensorflow==2.0.0) (0.2.0)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from
tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (2.23.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.6/dist-p
ackages (from tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (0.4.1)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-packages
(from tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (1.17.2)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from
tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (1.0.1)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.6/dist-packages (from
tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (50.3.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from
tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (3.2.2)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from keras-
applications>=1.0.8->tensorflow==2.0.0) (2.10.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0-
>tensorflow==2.0.0) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (3.0.4)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages
(from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (1.3.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages
(from google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in
/usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->ten
sorflow==2.0.0) (4.6)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages
(from google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0) (4.1.1)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in
/usr/local/lib/python3.6/dist-packages (from markdown>=2.6.8->tensorboard<2.1.0,>=2.0.0-
>tensorflow==2.0.0) (2.0.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from
```

```
requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.1.0,>=2.0.0-
>tensorflow==2.0.0) (3.1.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.6/dist-packages
(from pyasn1-modules>=0.2.1->google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->tensorflow==2.0.0)
(0.4.8)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from
importlib-metadata; python_version < "3.8"->markdown>=2.6.8->tensorboard<2.1.0,>=2.0.0-
>tensorflow==2.0.0) (3.2.0)
Building wheels for collected packages: gast
  Building wheel for gast (setup.py) ... done
  Created wheel for gast: filename=gast-0.2.2-cp36-none-any.whl size=7542
sha256=57b71d3bec51b0e014c2ac70316ff7a589ca3a1eaba91d90bd8098c8ff489e7f
  Stored in directory:
/root/.cache/pip/wheels/5c/2e/7e/a1d4d4fcebe6c381f378ce7743a3ced3699feb89bcfbdadadd
Successfully built gast
ERROR: tensorflowjs 2.5.0 has requirement tensorflow<3,>=2.1.0, but you'll have tensorflow 2.0.0 w
hich is incompatible.
ERROR: tensorflow-probability 0.11.0 has requirement gast>=0.3.2, but you'll have gast 0.2.2 which
is incompatible.
Installing collected packages: tensorflow-estimator, gast, tensorboard, keras-applications,
tensorflow
  Found existing installation: tensorflow-estimator 2.3.0
    Uninstalling tensorflow-estimator-2.3.0:
      Successfully uninstalled tensorflow-estimator-2.3.0
  Found existing installation: gast 0.3.3
    Uninstalling gast-0.3.3:
      Successfully uninstalled gast-0.3.3
  Found existing installation: tensorboard 2.3.0
    Uninstalling tensorboard-2.3.0:
      Successfully uninstalled tensorboard-2.3.0
  Found existing installation: tensorflow 2.3.0
    Uninstalling tensorflow-2.3.0:
      Successfully uninstalled tensorflow-2.3.0
Successfully installed gast-0.2.2 keras-applications-1.0.8 tensorboard-2.0.2 tensorflow-2.0.0 tens
orflow-estimator-2.0.1
```

In [4]:

```
import tensorflow as tf

print('\u2022 Using TensorFlow Version:', tf.__version__)
```

```
• Using TensorFlow Version: 2.0.0
```

Explore the Example Data

Let's start by downloading our example data, a .zip of 2,000 JPG pictures of cats and dogs, and extracting it locally in `/tmp`.

NOTE: The 2,000 images used in this exercise are excerpted from the "Dogs vs. Cats" dataset available on Kaggle, which contains 25,000 images. Here, we use a subset of the full dataset to decrease training time for educational purposes.

In [5]:

```
!wget --no-check-certificate \
  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
  -O /tmp/cats_and_dogs_filtered.zip
```

```
--2020-10-07 10:39:24--  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.20.128, 74.125.142.128,
74.125.195.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.20.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 68606236 (65M) [application/zip]
Saving to: '/tmp/cats_and_dogs_filtered.zip'

/tmp/cats_and_dogs_ 100%[===================>]  65.43M   147MB/s    in 0.4s

2020-10-07 10:39:25 (147 MB/s) - '/tmp/cats_and_dogs_filtered.zip' saved [68606236/68606236]
```

The following python code will use the OS library to use Operating System libraries, giving you access to the file system, and the zipfile library allowing you to unzip the data.

In [6]:

```
import os
import zipfile

local_zip = '/tmp/cats_and_dogs_filtered.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')

zip_ref.extractall('/tmp')
zip_ref.close()
```

The contents of the .zip are extracted to the base directory `/tmp/cats_and_dogs_filtered`, which contains `train` and `validation` subdirectories for the training and validation datasets (see the Machine Learning Crash Course for a refresher on training, validation, and test sets), which in turn each contain `cats` and `dogs` subdirectories.

In short: The training set is the data that is used to tell the neural network model that 'this is what a cat looks like', 'this is what a dog looks like' etc. The validation data set is images of cats and dogs that the neural network will not see as part of the training, so you can test how well or how badly it does in evaluating if an image contains a cat or a dog.

One thing to pay attention to in this sample: We do not explicitly label the images as cats or dogs. If you remember with the handwriting example earlier, we had labelled 'this is a 1', 'this is a 7' etc. Later you'll see something called an ImageGenerator being used -- and this is coded to read images from subdirectories, and automatically label them from the name of that subdirectory. So, for example, you will have a 'training' directory containing a 'cats' directory and a 'dogs' one. ImageGenerator will label the images appropriately for you, reducing a coding step.

Let's define each of these directories:

In [7]:

```
base_dir = '/tmp/cats_and_dogs_filtered'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Directory with our training cat/dog pictures
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with our validation cat/dog pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
```

Now, let's see what the filenames look like in the `cats` and `dogs` `train` directories (file naming conventions are the same in the `validation` directory):

In [8]:

```
train_cat_fnames = os.listdir( train_cats_dir )
train_dog_fnames = os.listdir( train_dogs_dir )

print(train_cat_fnames[:10])
print(train_dog_fnames[:10])
```

```
['cat.12.jpg', 'cat.870.jpg', 'cat.389.jpg', 'cat.115.jpg', 'cat.982.jpg', 'cat.366.jpg',
'cat.848.jpg', 'cat.266.jpg', 'cat.168.jpg', 'cat.983.jpg']
['dog.16.jpg', 'dog.450.jpg', 'dog.316.jpg', 'dog.351.jpg', 'dog.252.jpg', 'dog.37.jpg',
'dog.301.jpg', 'dog.753.jpg', 'dog.343.jpg', 'dog.801.jpg']
```

Let's find out the total number of cat and dog images in the `train` and `validation` directories:

In [9]:

```
print('total training cat images :', len(os.listdir(      train_cats_dir ) ))
print('total training dog images :', len(os.listdir(      train_dogs_dir ) ))
```

```
print('total validation cat images :', len(os.listdir( validation_cats_dir ) ))
print('total validation dog images :', len(os.listdir( validation_dogs_dir ) ))
```

```
total training cat images : 1000
total training dog images : 1000
total validation cat images : 500
total validation dog images : 500
```

For both cats and dogs, we have 1,000 training images and 500 validation images.

Now let's take a look at a few pictures to get a better sense of what the cat and dog datasets look like. First, we configure the `matplotlib` parameters:

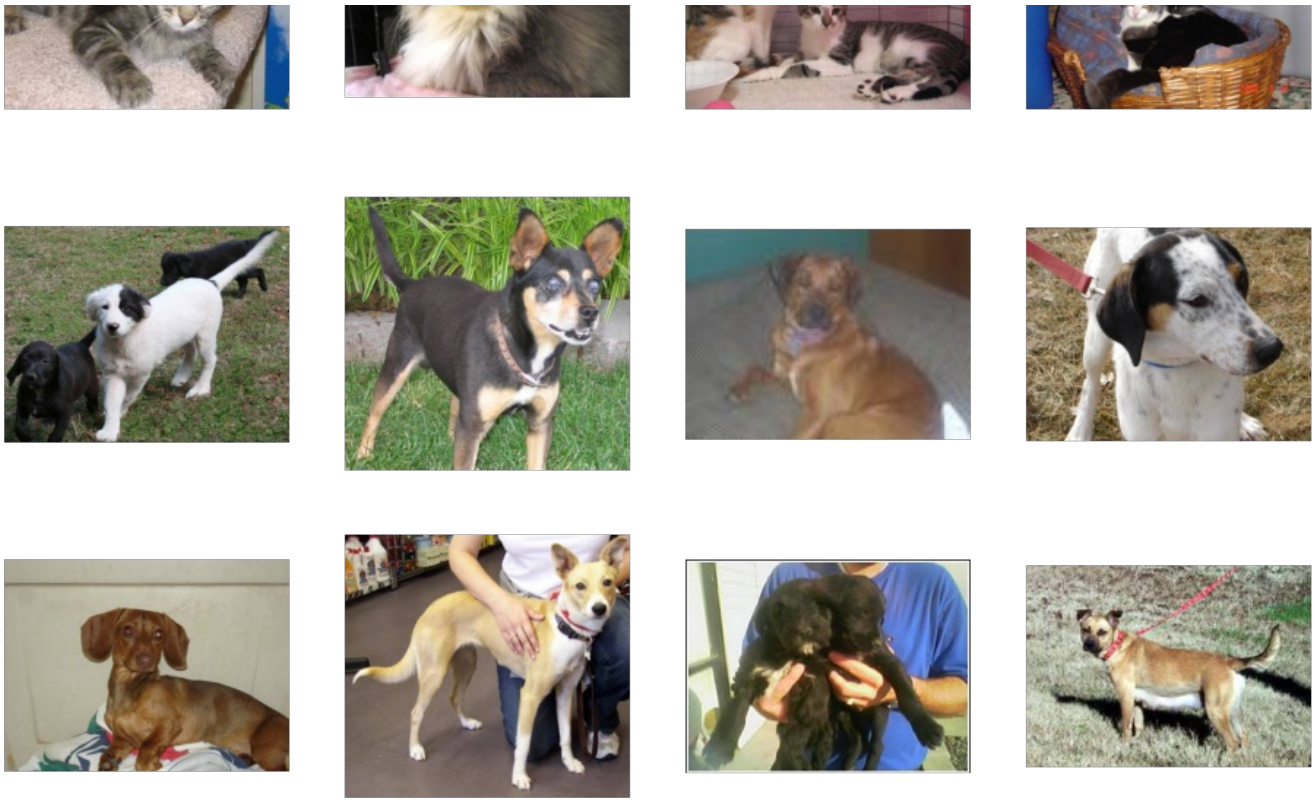
In [10]:

```
%matplotlib inline

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

pic_index = 0 # Index for iterating over images
```

Now, we display a batch of 8 cat and 8 dog pictures. You can re-run the cell to see a fresh batch each time:

In [11]:

```
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)

pic_index+=8

next_cat_pix = [os.path.join(train_cats_dir, fname)
                for fname in train_cat_fnames[ pic_index-8:pic_index]
               ]

next_dog_pix = [os.path.join(train_dogs_dir, fname)
                for fname in train_dog_fnames[ pic_index-8:pic_index]
               ]

for i, img_path in enumerate(next_cat_pix+next_dog_pix):
    # Set up subplot; subplot indices start at 1
    sp = plt.subplot(nrows, ncols, i + 1)
    sp.axis('Off') # Don't show axes (or gridlines)

    img = mpimg.imread(img_path)
    plt.imshow(img)

plt.show()
```



It may not be obvious from looking at the images in this grid, but an important note here, and a significant difference from the previous lesson is that these images come in all shapes and sizes. When you did the handwriting recognition example, you had 28x28 greyscale images to work with. These are color and in a variety of shapes. Before training a Neural network with them you'll need to tweak the images. You'll see that in the next section.

Ok, now that you have an idea for what your data looks like, the next step is to define the model that will be trained to recognize cats or dogs from these images

Building a Small Model from Scratch to Get to ~72% Accuracy

In the previous section you saw that the images were in a variety of shapes and sizes. In order to train a neural network to handle them you'll need them to be in a uniform size. We've chosen 150x150 for this, and you'll see the code that preprocesses the images to that shape shortly.

But before we continue, let's start defining the model. We will define a Sequential layer as before, adding some convolutional layers first. Note the input shape parameter this time. In the earlier example it was 28x28x1, because the image was 28x28 in greyscale (8 bits, 1 byte for color depth). This time it is 150x150 for the size and 3 (24 bits, 3 bytes) for the color depth.

We then add a couple of convolutional layers as in the previous example, and flatten the final result to feed into the densely connected layers.

Finally we add the densely connected layers.

Note that because we are facing a two-class classification problem, i.e. a *binary classification problem*, we will end our network with a *sigmoid* activation, so that the output of our network will be a single scalar between 0 and 1, encoding the probability that the current image is class 1 (as opposed to class 0).

In [12]:

```
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 bytes color
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
```

```
    # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('cats') and 1 fo
r the other ('dogs')
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

The `model.summary()` method call prints a summary of the NN

In [13]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 18496)             0
_________________________________________________________________
dense (Dense)                (None, 512)               9470464
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

The "output shape" column shows how the size of your feature map evolves in each successive layer. The convolution layers reduce the size of the feature maps by a bit due to padding, and each pooling layer halves the dimensions.

Next, we'll configure the specifications for model training. We will train our model with the `binary_crossentropy` loss, because it's a binary classification problem and our final activation is a sigmoid. (For a refresher on loss metrics, see the Machine Learning Crash Course.) We will use the `rmsprop` optimizer with a learning rate of `0.001`. During training, we will want to monitor classification accuracy.

NOTE: In this case, using the RMSprop optimization algorithm is preferable to stochastic gradient descent (SGD), because RMSprop automates learning-rate tuning for us. (Other optimizers, such as Adam and Adagrad, also automatically adapt the learning rate during training, and would work equally well here.)

In [14]:

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics = ['accuracy'])
```

Data Preprocessing

Let's set up data generators that will read pictures in our source folders, convert them to `float32` tensors, and feed them (with their labels) to our network. We'll have one generator for the training images and one for the validation images. Our generators will yield batches of 20 images of size 150x150 and their labels (binary).

As you may already know, data that goes into neural networks should usually be normalized in some way to make it more amenable to processing by the network. (It is uncommon to feed raw pixels into a convnet.) In our case, we will preprocess our images by normalizing the pixel values to be in the `[0, 1]` range (originally all values are in the `[0, 255]` range).

In Keras this can be done via the `keras.preprocessing.image.ImageDataGenerator` class using the `rescale` parameter. This `ImageDataGenerator` class allows you to instantiate generators of augmented image batches (and their labels) via `.flow(data, labels)` or `.flow_from_directory(directory)`. These generators can then be used with the Keras model methods that accept data generators as inputs: `fit`, `evaluate_generator`, and `predict_generator`.

In [15]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))
# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=20,
                                                         class_mode  = 'binary',
                                                         target_size = (150, 150))
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

Training

Let's train on all 2,000 images available, for 15 epochs, and validate on all 1,000 test images. (This may take a few minutes to run.)

Do note the values per epoch.

You'll see 4 values per epoch -- Loss, Accuracy, Validation Loss and Validation Accuracy.

The Loss and Accuracy are a great indication of progress of training. It's making a guess as to the classification of the training data, and then measuring it against the known label, calculating the result. Accuracy is the portion of correct guesses. The Validation accuracy is the measurement with the data that has not been used in training. As expected this would be a bit lower. You'll learn about why this occurs in the section on overfitting later in this course.

In [16]:

```
history = model.fit(train_generator,
                              validation_data=validation_generator,
                              steps_per_epoch=100,
                              epochs=15,
                              validation_steps=50,
                              verbose=2)
```

```
Train for 100 steps, validate for 50 steps
Epoch 1/15
100/100 - 64s - loss: 0.7285 - accuracy: 0.5515 - val_loss: 0.6661 - val_accuracy: 0.6510
Epoch 2/15
100/100 - 63s - loss: 0.6371 - accuracy: 0.6430 - val_loss: 0.5820 - val_accuracy: 0.6880
Epoch 3/15
100/100 - 64s - loss: 0.5373 - accuracy: 0.7325 - val_loss: 0.5453 - val_accuracy: 0.7370
Epoch 4/15
100/100 - 63s - loss: 0.4647 - accuracy: 0.7850 - val_loss: 0.5723 - val_accuracy: 0.7030
Epoch 5/15
100/100 - 63s - loss: 0.3833 - accuracy: 0.8330 - val_loss: 0.6389 - val_accuracy: 0.7100
Epoch 6/15
100/100 - 63s - loss: 0.2947 - accuracy: 0.8810 - val_loss: 0.7956 - val_accuracy: 0.6810
Epoch 7/15
100/100 - 63s - loss: 0.2142 - accuracy: 0.9155 - val_loss: 0.8384 - val_accuracy: 0.6990
Epoch 8/15
100/100 - 64s - loss: 0.1554 - accuracy: 0.9360 - val_loss: 1.0533 - val_accuracy: 0.7080
Epoch 9/15
100/100 - 64s - loss: 0.1132 - accuracy: 0.9565 - val_loss: 0.9798 - val_accuracy: 0.7110
```

```
Epoch 10/15
100/100 - 64s - loss: 0.0766 - accuracy: 0.9730 - val_loss: 1.1497 - val_accuracy: 0.7100
Epoch 11/15
100/100 - 64s - loss: 0.0754 - accuracy: 0.9745 - val_loss: 1.3253 - val_accuracy: 0.7170
Epoch 12/15
100/100 - 64s - loss: 0.0638 - accuracy: 0.9790 - val_loss: 1.4159 - val_accuracy: 0.7120
Epoch 13/15
100/100 - 64s - loss: 0.0830 - accuracy: 0.9745 - val_loss: 1.4690 - val_accuracy: 0.6980
Epoch 14/15
100/100 - 64s - loss: 0.0579 - accuracy: 0.9875 - val_loss: 1.6677 - val_accuracy: 0.7100
Epoch 15/15
100/100 - 64s - loss: 0.0458 - accuracy: 0.9860 - val_loss: 2.0899 - val_accuracy: 0.6970
```

Evaluating Accuracy and Loss for the Model

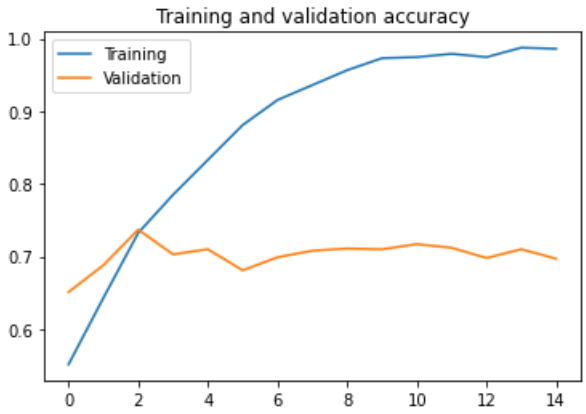
Let's plot the training/validation accuracy and loss as collected during training:

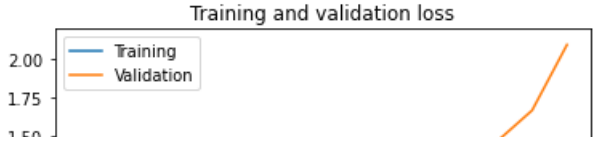
In [17]:

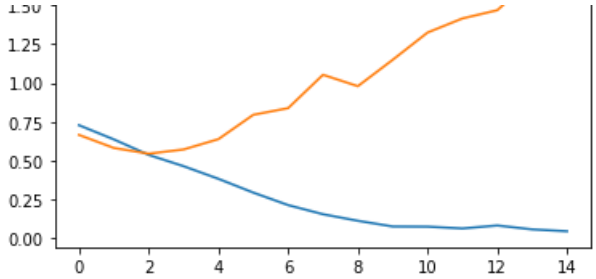
```
#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
acc      = history.history[     'accuracy' ]
val_acc  = history.history[ 'val_accuracy' ]
loss     = history.history[    'loss' ]
val_loss = history.history['val_loss' ]

epochs   = range(len(acc)) # Get number of epochs

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot  ( epochs,     acc, label='Training')
plt.plot  ( epochs, val_acc, label='Validation')
plt.title ('Training and validation accuracy')
plt.legend()
plt.figure()

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot  ( epochs,     loss, label='Training')
plt.plot  ( epochs, val_loss, label='Validation')
plt.legend()
plt.title ('Training and validation loss')
```

Out[17]:

```
Text(0.5, 1.0, 'Training and validation loss')
```







As you can see, we are **overfitting** like it's getting out of fashion. Our training accuracy (in blue) gets close to 100% (!) while our validation accuracy (in orange) stalls as 70%. Our validation loss reaches its minimum after only five epochs.

Since we have a relatively small number of training examples (2000), overfitting should be our number one concern. Overfitting happens when a model exposed to too few examples learns patterns that do not generalize to new data, i.e. when the model starts using irrelevant features for making predictions. For instance, if you, as a human, only see three images of people who are lumberjacks, and three images of people who are sailors, and among them the only person wearing a cap is a lumberjack, you might start thinking that wearing a cap is a sign of being a lumberjack as opposed to a sailor. You would then make a pretty lousy lumberjack/sailor classifier.

Overfitting is the central problem in machine learning: given that we are fitting the parameters of our model to a given dataset, how can we make sure that the representations learned by the model will be applicable to data never seen before? How do we avoid learning things that are specific to the training data?

In the next exercise, we'll look at ways to prevent overfitting in the cat vs. dog classification model.

Save the Model

In the cell below, save the trained model as a Keras model (`.h5` file).

HINT: Use `model.save()`. Feel free to take a look at the `Linear-to-JavaScript.ipynb` example.

In [18]:

```
# EXERCISE: Save the trained model as a Keras HDF5 file.

saved_model_path = "./my_model.h5"

# YOUR CODE HERE
model.save(saved_model_path)
```

Run the TensorFlow.js Converter on The Saved Keras Model

In the cell below, use the `tensorflowjs_converter` to convert the saved Keras model into JSON format.

HINT: Make sure you specify the format of the input model as Keras by using the `--input_format` option. Feel free to take a look at the `Linear-to-JavaScript.ipynb` example and the TensorFlow.js converter documentation.

In [19]:

```
# EXERCISE: Use the tensorflow.js converter to convert the saved Keras model into JSON format.

# YOUR CODE HERE
!tensorflowjs_converter --input_format=keras {saved_model_path} ./
```

If you did things correctly, you should now have a **JSON** file named `model.json` and various `.bin` files, such as `group1-shard1of10.bin`. The number of `.bin` files will depend on the size of your model: the larger your model, the greater the number of `.bin` files. The `model.json` file contains the architecture of your model and the `.bin` files will contain the weights of your model.