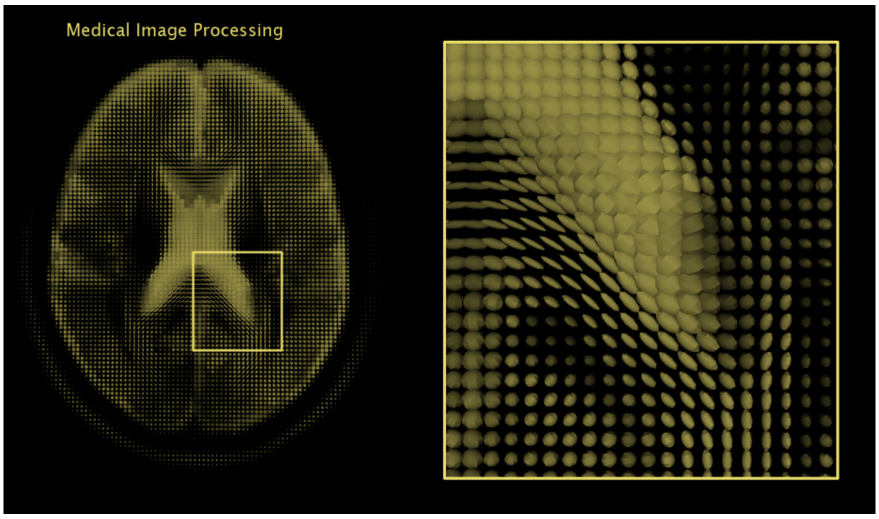
Taposh Dutta-Roy
May 9 · 7 min read

Medical Image Analysis with Deep Learning —III



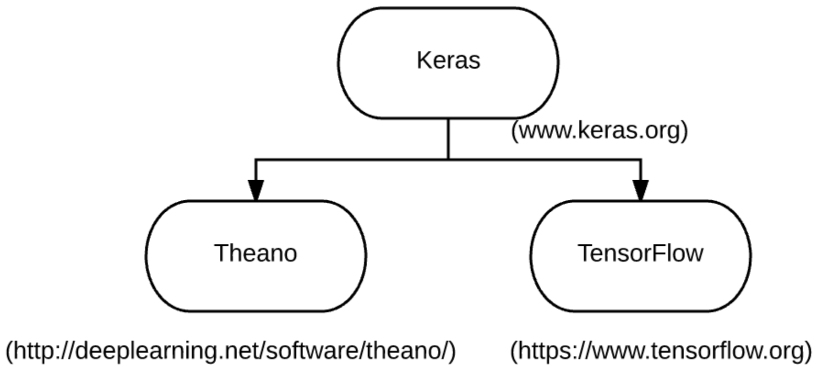
In the last article we will talk about basics of deep learning from the lens of Convolutional Neural Nets. In this article we will focus—basic deep learning using Keras and Theano. We will do 2 examples one using keras for basic predictive analytics and other a simple example of image analysis using VGG.

I have realized that this topic is broad and deep and will need a few more articles. In the next few articles we will discuss difference between DICOM and NIFTI formats for medical imaging , expand our learning further and discuss how to use deep learning for 2D lung segmentation analysis. We then move to analyze 3D lung segmentation. We will also discuss how medical image analysis was done prior deep learning and how we can do it now. I would also like to welcome and thank my new partners who will help me with putting this all together—Flavio Trolese, Partner at 4Quant, Kevin Mader, Co-founder of 4Quant and Lecturer at ETH Zurich and Cyriac Joshy.

In this article we will discuss Keras and use two examples one showing how to use keras for simple predictive analysis tasks and other doing a image analysis.

What is Keras?

From the Keras website—Keras is a deep learning library for Theanos and Tensor flow.


Keras
(www.keras.org)
Theano
TensorFlow
(http://deeplearning.net/software/theano/)
(https://www.tensorflow.org)

Keras api running on top of theano and tensorflow

Keras is a high-level neural networks API, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation.

What is Theano and Tensor flow?

Theano as stated by Dr. James Bergstra et.al. at Scipy 2010, is a CPU and GPU math expression compiler. It is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano is a joint work done by some high profile researchers such as Yoshua Bengio and others Montereal Institute for Learning Algorithms (MILA). A very good tutorial on Theano at scipy 2010 is here. The picture below shows a comparison of Theano on GPU and CPU vs a few other tools as of 2010. This was published in the original paper "Theano: A CPU and GPU Math Compiler in Python"

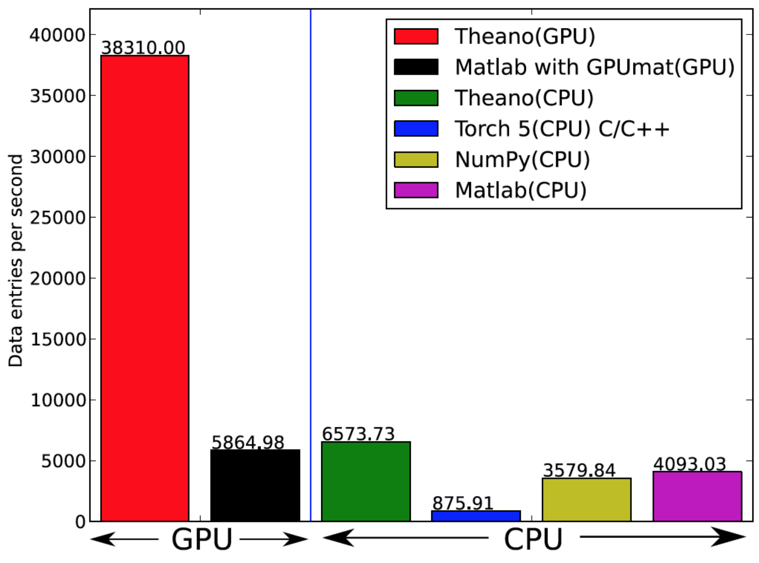


Figure 5: *Fitting a multi-layer perceptron to simulated data with various implementations of stochastic gradient descent. These models have 784 inputs, 500 hidden units, a 10-way classification, and are trained 60 examples at a time.*

Theano: A CPU and GPU Math Compiler in Python
James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, Yoshua Bengio

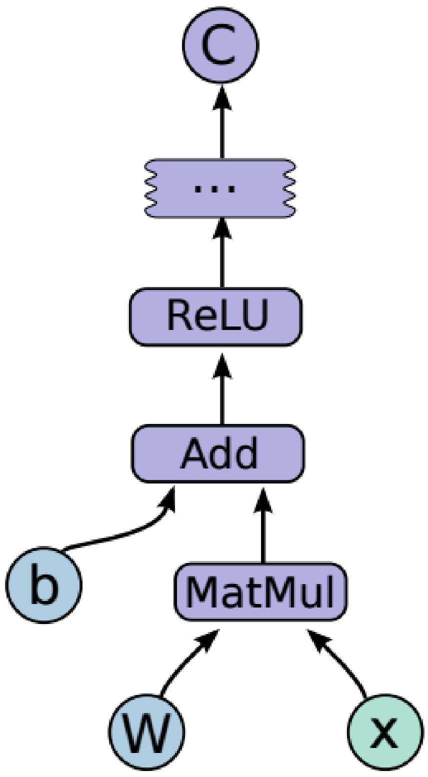
There are several other libraries built on top of Theano, including Pylearn2 and GroundHog (also developed by MILA), Lasagne, and Blocks and Fuel.

SciPy 2010 - James Bergstra - Transparent GPU Computing with Theano : Free Download & Streaming ...

SciPy 2010 Day 1 Main Track 4 James Bergstra, Olivier Breuleux, Frederic Bastien, Pascal Lamblin, Razvan Pascanu...

archive.org

TensorFlow was developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization. It was developed for conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. TensorFlow™, as stated from their website is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. Visually the code would like the figure shown below [Source]



TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems

Example of Predictive Analytics with Keras

In this example we will use Sonar data set from UCI website to do a simple predictive model. In the code below, we get data directly from the UCI website and split it in 60:40 ratio for training vs testing. We use keras for predictive modeling and sklearn for label encoding.

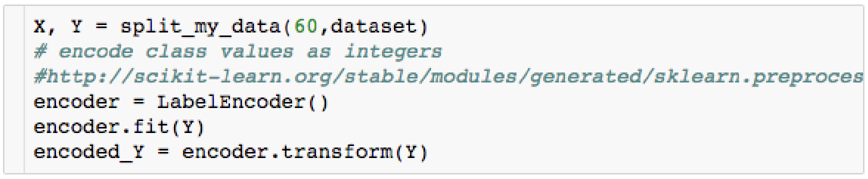
```python
import pandas as pd
import urllib.request as ur
from tqdm import tqdm
import io
import requests
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.preprocessing import LabelEncoder

#Read csv file from UCI Website
def read_csv_from_url(url_data):
    uci_dataset = url_data
    s=requests.get(uci_dataset).content
    the_dataframe=pd.read_csv(io.StringIO(s.decode('utf-8')))
    dataset = the_dataframe.values
    return dataset

# Random seed for reproducibility
seed_val = 7
numpy.random.seed(seed_val)

# split into input (X) and output (Y) variables
def split_my_data(ratio,dataset):
    X = dataset[:,0:ratio].astype(float)
    Y = dataset[:,ratio]
    return X, Y
```

In the next snippet of code we read the data-set and see the data using the functions we defined above. We print the data-set and find out that our dependent variable needs encoding.

```python
url_sonar="https://archive.ics.uci.edu/ml/machine-learning-database
dataset = read_csv_from_url(url_sonar)
print(dataset)
```

```
[[0.0453 0.0523 0.0843 ..., 0.0052 0.0044 'R']
 [0.0262 0.0582 0.1099 ..., 0.0095 0.0078 'R']
 [0.01 0.0171 0.0623 ..., 0.004 0.0117 'R']
 ...,
 [0.0522 0.0437 0.018 ..., 0.0077 0.0031 'M']
 [0.0303 0.0353 0.049 ..., 0.0036 0.0048 'M']
 [0.026 0.0363 0.0136 ..., 0.0061 0.0115 'M']]
```

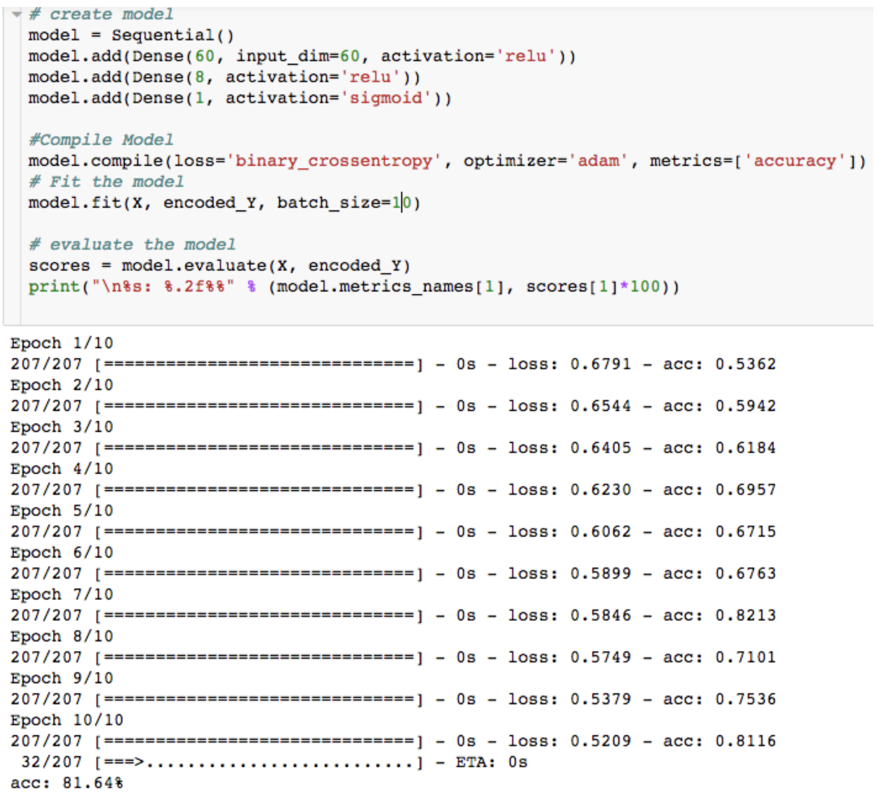
We use LabelEncoder from scikit-learn to do label encoding to covert R and M into numbers 0 and 1. The generic process of doing this is called one-hot encoding. One hot encoding transforms categorical features to a format that works better with algorithms. In this example our Y variable was categorical with values “R” and “M”. Using the Label encoder we converted this into “1” and “0”.



```
X, Y = split_my_data(60,dataset)
# encode class values as integers
#http://scikit-learn.org/stable/modules/generated/sklearn.preproces
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
```

Label encoder from Scikit-learn

We then create a model using Keras.



```
# create model
model = Sequential()
model.add(Dense(60, input_dim=60, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#Compile Model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, encoded_Y, batch_size=10)

# evaluate the model
scores = model.evaluate(X, encoded_Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
Epoch 1/10
207/207 [==============================] - 0s - loss: 0.6791 - acc: 0.5362
Epoch 2/10
207/207 [==============================] - 0s - loss: 0.6544 - acc: 0.5942
Epoch 3/10
207/207 [==============================] - 0s - loss: 0.6405 - acc: 0.6184
Epoch 4/10
207/207 [==============================] - 0s - loss: 0.6230 - acc: 0.6957
Epoch 5/10
207/207 [==============================] - 0s - loss: 0.6062 - acc: 0.6715
Epoch 6/10
207/207 [==============================] - 0s - loss: 0.5899 - acc: 0.6763
Epoch 7/10
207/207 [==============================] - 0s - loss: 0.5846 - acc: 0.8213
Epoch 8/10
207/207 [==============================] - 0s - loss: 0.5749 - acc: 0.7101
Epoch 9/10
207/207 [==============================] - 0s - loss: 0.5379 - acc: 0.7536
Epoch 10/10
207/207 [==============================] - 0s - loss: 0.5209 - acc: 0.8116
 32/207 [===>..........................] - ETA: 0s
acc: 81.64%
```

> *The accuracy using simplistic model without any pre-processing is 81.64%*

Example Image Analysis with Keras

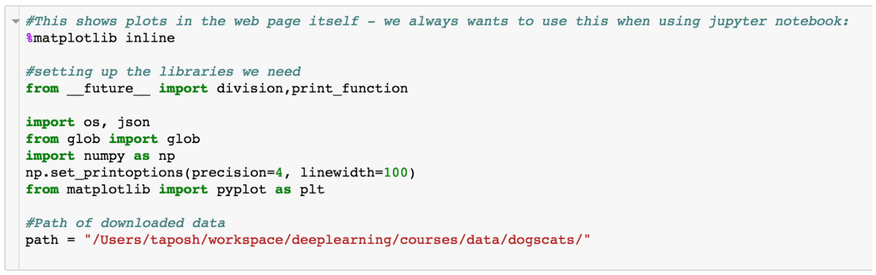
In order to explain image processing with keras, we will use data from Kaggle competition—dogs and cats. The goal of this competition is to develop an algorithm to classify whether images contain either a dog or a cat. This task is easy for humans, dogs, and cats but not for computers. In this challenge there are 25,000 labelled dog and cat photos available for training, and 12,500 in the test set that we have to try to label for this competition. According to the Kaggle web-site, when this competition was launched (end of 2013):

"***State of the art****: The current literature suggests machine classifiers can score above 80% accuracy on this task*". So if we can beat 80%, then we will be at the cutting edge as at 2013!

I highly recommend MOOC from Fast.ai for more details, learning next steps and cutting edge research on the deep-learning. I have referred fast.ai for the code below as well as it serves an excellent starting point.

Step 1: Setting it up

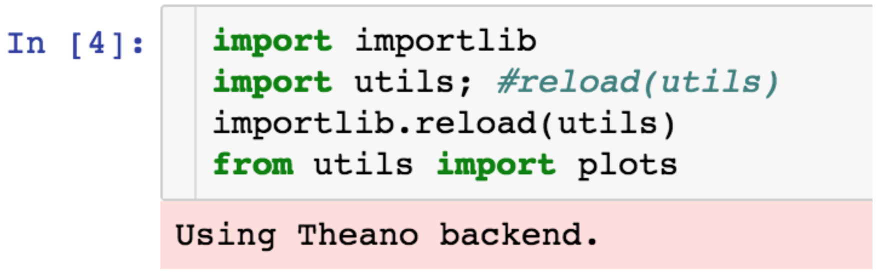
Download the data from Kaggle website for the dogs and cats, and save it on your laptop. For the example in this article I am running the code on my mac.



```
#This shows plots in the web page itself - we always wants to use this when using jupyter notebook:
%matplotlib inline

#setting up the libraries we need
from __future__ import division,print_function

import os, json
from glob import glob
import numpy as np
np.set_printoptions(precision=4, linewidth=100)
from matplotlib import pyplot as plt

#Path of downloaded data
path = "/Users/taposh/workspace/deeplearning/courses/data/dogscats/"
```

Basic Set-up

Jeremy Howard, in his class has provided a utility python file that helps to get the basic functions encapsulated. For the initial part we will use this utility file. The file is available here. As we go into more details we will unpack this file and see whats behind it.

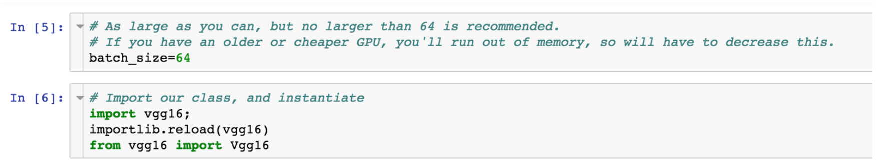


```
In [4]: import importlib
        import utils; #reload(utils)
        importlib.reload(utils)
        from utils import plots

Using Theano backend.
```

Step 2: Using VGG

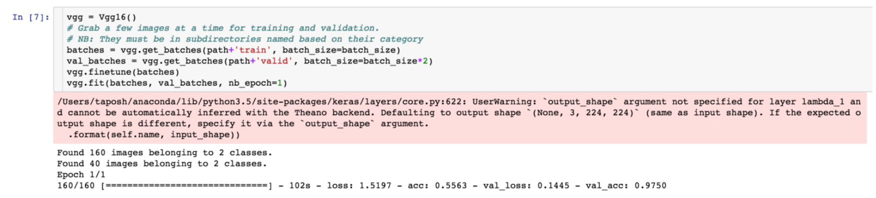
Our first step is simply to use a model that has been fully created for us, which can recognize a wide variety (1,000 categories) of images. We will use 'VGG', which won the 2014 Imagenet competition, and is a very simple model to create and understand. The VGG Imagenet team created both a larger, slower, slightly more accurate model (*VGG 19*) and a smaller, faster model (*VGG 16*). We will be using VGG 16 since the much slower performance of VGG19 is generally not worth the very minor improvement in accuracy.

We have created a python class, *Vgg16*, which makes using the VGG 16 model very straightforward. Vgg16 is also available from fast.ai's github details are here.



```
In [5]: # As large as you can, but no larger than 64 is recommended.
        # If you have an older or cheaper GPU, you'll run out of memory, so will have to decrease this.
        batch_size=64

In [6]: # Import our class, and instantiate
        import vgg16;
        importlib.reload(vgg16)
        from vgg16 import Vgg16
```

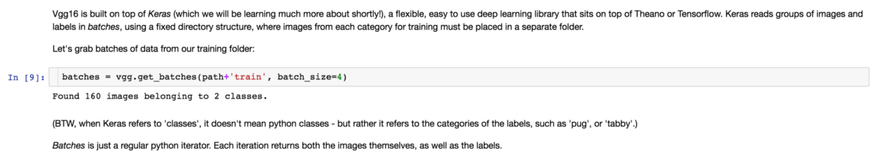
Step 3: Instantiate VGG



```
In [7]: vgg = Vgg16()
        # Grab a few images at a time for training and validation.
        # NB: They must be in subdirectories named based on their category
        batches = vgg.get_batches(path+'train', batch_size=batch_size)
        val_batches = vgg.get_batches(path+'valid', batch_size=batch_size*2)
        vgg.finetune(batches)
        vgg.fit(batches, val_batches, nb_epoch=1)

/Users/taposh/anaconda/lib/python3.5/site-packages/keras/layers/core.py:622: UserWarning: `output_shape` argument not specified for layer lambda_1 an
d cannot be automatically inferred with the Theano backend. Defaulting to output shape `(None, 3, 224, 224)` (same as input shape). If the expected o
utput shape is different, specify it via the `output_shape` argument.
  .format(self.name, input_shape))

Found 160 images belonging to 2 classes.
Found 40 images belonging to 2 classes.
Epoch 1/1
160/160 [==============================] - 102s - loss: 1.5197 - acc: 0.5563 - val_loss: 0.1445 - val_acc: 0.9750
```

Vgg16 is built on top of *Keras* (which we will be learning much more about shortly!), a flexible, easy to use deep learning library that sits on top of Theano or Tensorflow. Keras reads groups of images and labels in *batches*, using a fixed directory structure, where images from each category for training must be placed in a separate folder.

Let's grab batches of data from our training folder:



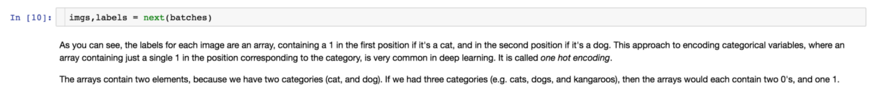
Vgg16 is built on top of *Keras* (which we will be learning much more about shortly!), a flexible, easy to use deep learning library that sits on top of Theano or Tensorflow. Keras reads groups of images and labels in *batches*, using a fixed directory structure, where images from each category for training must be placed in a separate folder.

Let's grab batches of data from our training folder:

```
In [9]:  batches = vgg.get_batches(path+'train', batch_size=4)
         Found 160 images belonging to 2 classes.
```

(BTW, when Keras refers to 'classes', it doesn't mean python classes - but rather it refers to the categories of the labels, such as 'pug', or 'tabby'.)

Batches is just a regular python iterator. Each iteration returns both the images themselves, as well as the labels.

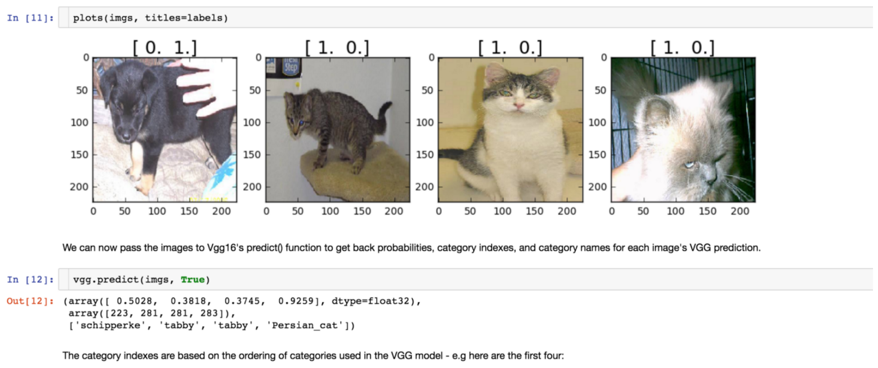


```
In [10]:  imgs,labels = next(batches)
```

As you can see, the labels for each image are an array, containing a 1 in the first position if it's a cat, and in the second position if it's a dog. This approach to encoding categorical variables, where an array containing just a single 1 in the position corresponding to the category, is very common in deep learning. It is called *one hot encoding*.

The arrays contain two elements, because we have two categories (cat, and dog). If we had three categories (e.g. cats, dogs, and kangaroos), then the arrays would each contain two 0's, and one 1.

Step 4: Predict Dog vs Cats



```
In [11]:  plots(imgs, titles=labels)
```

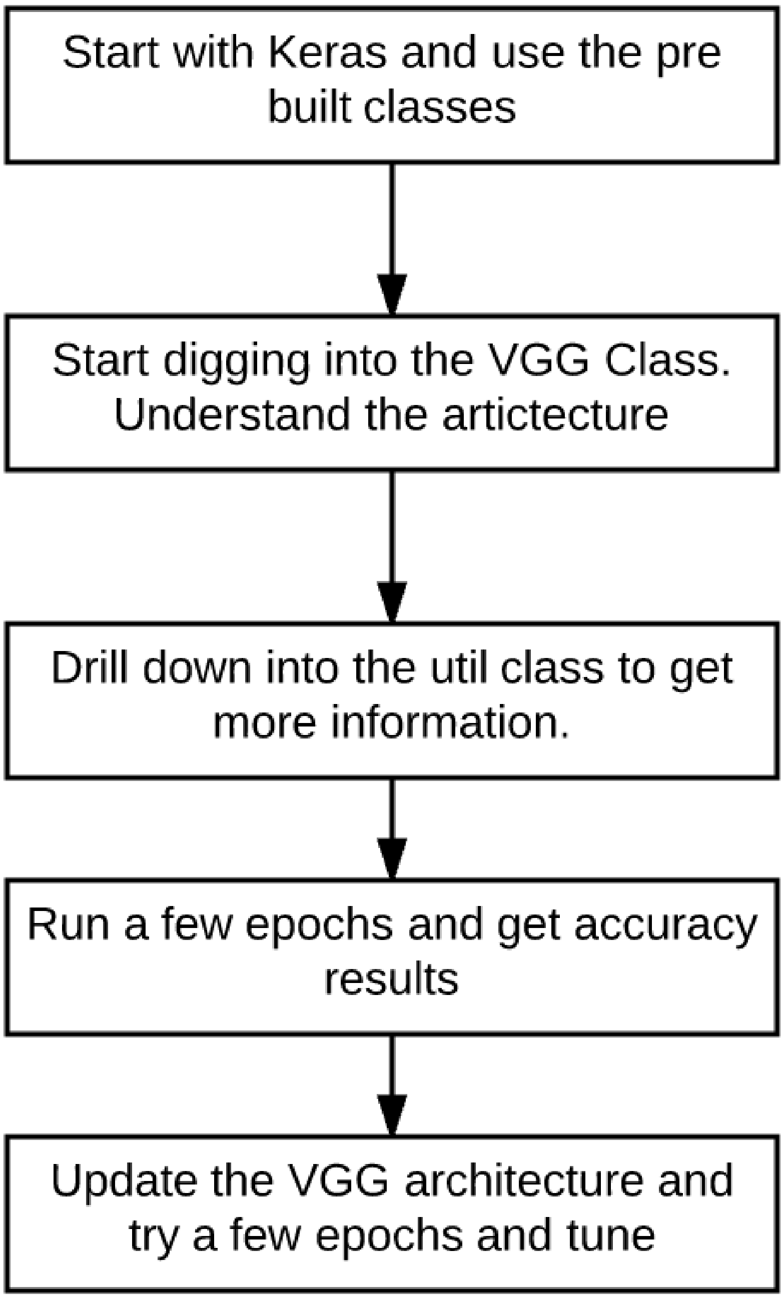
We can now pass the images to Vgg16's predict() function to get back probabilities, category indexes, and category names for each image's VGG prediction.

```
In [12]:  vgg.predict(imgs, True)
Out[12]: (array([ 0.5028,  0.3818,  0.3745,  0.9259], dtype=float32),
          array([223, 281, 281, 283]),
          ['schipperke', 'tabby', 'tabby', 'Persian_cat'])
```

The category indexes are based on the ordering of categories used in the VGG model - e.g here are the first four:

Step 5: Summing it up and code files…

To sum this up for this article, the approach for dogs and cats classification that I recommend is —


Start with Keras and use the pre built classes
Start digging into the VGG Class. Understand the artictecture
Drill down into the util class to get more information.
Run a few epochs and get accuracy results
Update the VGG architecture and try a few epochs and tune

Conclusion

If you are with me this far, then you have taken the theory we discussed in last article and done some practical programming. If you follow the instructions above and do the two examples, you have done your first predictive model using Keras and done image analysis as well. Due to the length of the code, I have not discussed the details in here, but have linked them. If you review the links and have questions, please feel to reach out to me or fast.ai (for dogs-cats image analysis)

References:

Keras Documentation

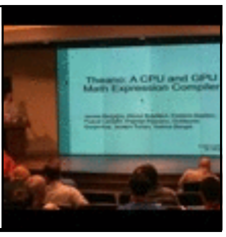
Keras: Deep Learning library for Theano and TensorFlow

keras.io

Welcome - Theano 0.9.0 documentation

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi...

deeplearning.net



Develop Your First Neural Network in Python With Keras Step-By-Step - Machine Learning Mastery

Keras is a powerful easy-to-use Python library for developing and evaluating deep learning models. It wraps the...

machinelearningmastery.com

http://svn.ucc.asn.au:8080/oxinabox/Uni%20Notes/honours/Background%20Reading/theano_scipy2010.pdf

One Hot Encoding for Machine learning

Many learning algorithms either learn a single weight per feature, or they use distances between samples. The former is...

stackoverflow.com

https://github.com/fastai/courses/blob/master/deeplearning1/nbs/utils.py

https://github.com/GUR9000/Deep_MRI_brain_extraction