Artificial Intelligence Nanodegree

Convolutional Neural Networks

Project: Write an Algorithm for a Dog Identification App

In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'(IMPLEMENTATION)'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section, and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the iPython Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

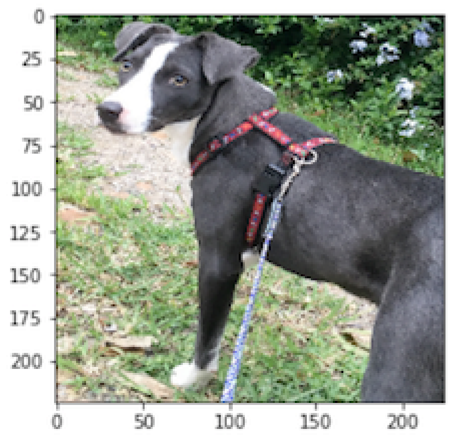
> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

The rubric contains *optional* "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. If you decide to pursue the "Stand Out Suggestions", you should include the code in this IPython notebook.

Why We're Here

In this notebook, you will make the first steps towards developing an algorithm that could be used as part of a mobile or web app. At the end of this project, your code will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling. The image below displays potential sample output of your finished project (... but we expect that each student's algorithm will behave differently!).

```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```



In this real-world setting, you will need to piece together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed. There are many points of possible failure, and no perfect algorithm exists. Your imperfect solution will nonetheless create a fun user experience!

The Road Ahead

We break the notebook into separate steps. Feel free to use the links below to navigate the notebook.

- Step 0: Import Datasets
- Step 1: Detect Humans
- Step 2: Detect Dogs
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
- Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 6: Write your Algorithm
- Step 7: Test Your Algorithm

Step 0: Import Datasets

Import Dog Dataset

In the code cell below, we import a dataset of dog images. We populate a few variables through the use of the `load_files` function from the scikit-learn library:

- `train_files`, `valid_files`, `test_files` - numpy arrays containing file paths to images
- `train_targets`, `valid_targets`, `test_targets` - numpy arrays containing onehot-encoded classification labels
- `dog_names` - list of string-valued dog breed names for translating labels

In [1]:
```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from glob import glob

# define function to load train, test, and validation datasets
def load_dataset(path):
    data = load_files(path)
    dog_files = np.array(data['filenames'])
    dog_targets = np_utils.to_categorical(np.array(data['target']), 133)
    return dog_files, dog_targets

# load train, test, and validation datasets
train_files, train_targets = load_dataset('dogImages/train')
valid_files, valid_targets = load_dataset('dogImages/valid')
test_files, test_targets = load_dataset('dogImages/test')

# load list of dog names
dog_names = [item[20:-1] for item in sorted(glob("dogImages/train/*/"))]

# print statistics about the dataset
print('There are %d total dog categories.' % len(dog_names))
print('There are %s total dog images.\n' % len(np.hstack([train_files, valid_f
iles, test_files])))
print('There are %d training dog images.' % len(train_files))
print('There are %d validation dog images.' % len(valid_files))
print('There are %d test dog images.'% len(test_files))
```

```
Using TensorFlow backend.

There are 133 total dog categories.
There are 8351 total dog images.

There are 6680 training dog images.
There are 835 validation dog images.
There are 836 test dog images.
```

Import Human Dataset

In the code cell below, we import a dataset of human images, where the file paths are stored in the numpy array `human_files`.

In [2]:
```
import random
random.seed(8675309)

# load filenames in shuffled human dataset
human_files = np.array(glob("lfw/*/*"))
random.shuffle(human_files)

# print statistics about the dataset
print('There are %d total human images.' % len(human_files))
```

```
There are 13233 total human images.
```

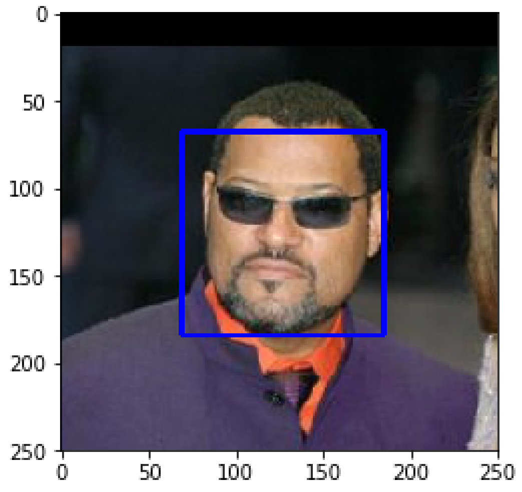
Step 1: Detect Humans

We use OpenCV's implementation of Haar feature-based cascade classifiers (http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html) to detect human faces in images. OpenCV provides many pre-trained face detectors, stored as XML files on github (https://github.com/opencv/opencv/tree/master/data/haarcascades). We have downloaded one of these detectors and stored it in the `haarcascades` directory.

In the next code cell, we demonstrate how to use this detector to find human faces in a sample image.

In [3]:
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_al
t.xml')

# load color (BGR) image
img = cv2.imread(human_files[3])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

```
Number of faces detected: 1
```



Before using any of the face detectors, it is standard procedure to convert the images to grayscale. The `detectMultiScale` function executes the classifier stored in `face_cascade` and takes the grayscale image as a parameter.

In the above code, `faces` is a numpy array of detected faces, where each row corresponds to a detected face. Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. The first two entries in the array (extracted in the above code as `x` and `y`) specify the horizontal and vertical positions of the top left corner of the bounding box. The last two entries in the array (extracted here as `w` and `h`) specify the width and height of the box.

Write a Human Face Detector

We can use this procedure to write a function that returns `True` if a human face is detected in an image and `False` otherwise. This function, aptly named `face_detector`, takes a string-valued file path to an image as input and appears in the code block below.

In [4]:
```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

In [5]:
```
# returns "True" if face is detected in image stored at img_path
def face_detector2(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray,scaleFactor=1.2)
    return len(faces) > 0
```

(IMPLEMENTATION) Assess the Human Face Detector

Question 1: Use the code cell below to test the performance of the `face_detector` function.

- What percentage of the first 100 images in `human_files` have a detected human face?
- What percentage of the first 100 images in `dog_files` have a detected human face?

Ideally, we would like 100% of human images with a detected face and 0% of dog images with a detected face. You will see that our algorithm falls short of this goal, but still gives acceptable performance. We extract the file paths for the first 100 images from each of the datasets and store them in the numpy arrays `human_files_short` and `dog_files_short`.

Answer:

From the OpenCV face detector, i got the following accuracy:

human correct = 99%

dog correct (not human) = 89 %

I created 2 more human face detectors:

```
1  --  The first one was a small modification to the OpenCV, just add a scaleFactor
=1.2, this increased the accuracy of the OpenCV to:


    human correct = 99%

    dog correct (not human) = 95 %




2 --  I implemented a Binary Classification Deep CNN for human detection.  I got th
e following accuracy:



    Accuracy using human_test_files: 100.0 %

    Accuracy using (dog) test_files: 100.0 %

    human correct = 100.0 %

    dog correct(not human) =  98.0 %
```

In [6]:
```
human_files_short = human_files[:100]
dog_files_short = train_files[:100]
# Do NOT modify the code above this line.

## TODO: Test the performance of the face_detector algorithm
## on the images in human_files_short and dog_files_short.

human_detection=100*sum(1 for face in human_files_short if
face_detector(face))/len(human_files_short)
dog_detection=100*(1-sum(1 for dogface in dog_files_short if face_detector(dog
face))/len(dog_files_short))

print("human correct %=",human_detection)
print("dog correct %=",dog_detection)
```

```
human correct %= 99.0
dog correct %= 89.0
```

Question 2: This algorithmic choice necessitates that we communicate to the user that we accept human images only when they provide a clear view of a face (otherwise, we risk having unneccessarily frustrated users!). In your opinion, is this a reasonable expectation to pose on the user? If not, can you think of a way to detect humans in images that does not necessitate an image with a clearly presented face?

Answer:

A binary classification Deep CNN was created for face/dog detection, with the following accuracies:

```
Accuracy using human_test_files: 100.0 %

Accuracy using (dog) test_files: 100.0 %

human correct = 100.0 %

dog correct(not human) =  98.0 %
```

In [7]:
```
## (Optional) TODO: Report the performance of another
## face detection algorithm on the LFW dataset
### Feel free to use as many code cells as needed.
human_detection=100*sum(1 for face in human_files_short if
face_detector2(face))/len(human_files_short)
dog_detection=100*(1-sum(1 for dogface in dog_files_short if face_detector2(do
gface))/len(dog_files_short))

print("human correct %=",human_detection)
print("dog correct %=",dog_detection)
```

```
human correct %= 99.0
dog correct %= 95.0
```

CNN For Dog/Human Detection

Create list of tensors for dog/human CNN

In [9]:
```
from keras.preprocessing import image
from tqdm import tqdm
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True


def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D
 tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in
tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

Create data and labels for CNN binary classification

In [10]:
```
human_train_files=human_files[:len(train_files)]
human_test_files=human_files[len(train_files)+1:len(train_files)+len(test_file
s)+1]

dog_label_vectors=np.zeros(len(train_files))
dog_label_test=np.zeros(len(test_files))
human_label_vectors=np.ones(len(human_train_files))
human_label_test=np.ones(len(human_test_files))

train_tensors_files=np.concatenate((train_files,human_train_files))
test_tensors_files=np.concatenate((test_files,human_test_files))

train_labels=np.concatenate([dog_label_vectors,human_label_vectors])
test_labels=np.concatenate([dog_label_test,human_label_test])

print("final train images:",len(train_tensors_files))
print("final train labels:",len(train_labels))

print("final test images:",len(test_tensors_files))
print("final test labels:",len(test_labels))

print("train_tensors_shape:",train_tensors_files.shape)

#train_labels2 = np.array([0] * len(trin_dog_tensors) + [1] * len(train_human_
tensors))

train_tensors = paths_to_tensor(train_tensors_files).astype('float32')/255
test_tensors = paths_to_tensor(test_tensors_files).astype('float32')/255
```

```
final train images: 13360
final train labels: 13360
final test images: 1672
final test labels: 1672
train_tensors_shape: (13360,)
```

```
100%|████████████████████████████████████████████████████████████████████████
████████████████████| 13360/13360 [06:29<00:00, 34.33it/s]
100%|████████████████████████████████████████████████████████████████████████
██████████████████████| 1672/1672 [00:44<00:00, 20.32it/s]
```

Create binary human CNN model

In [11]:
```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, BatchNo
rmalization, Activation
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

# 100%,100%,100%,98%
human_model = Sequential()
human_model.add(Conv2D(filters=16, kernel_size=2, padding='same',
activation='relu',input_shape=(224, 224, 3)))
human_model.add(MaxPooling2D(pool_size=2))
human_model.add(Conv2D(filters=32, kernel_size=3,padding='same', activation='r
elu'))
human_model.add(MaxPooling2D(pool_size=2))
human_model.add(Conv2D(filters=64, kernel_size=3, padding='same',
activation='relu'))
human_model.add(MaxPooling2D(pool_size=2))
human_model.add(Conv2D(filters=128, kernel_size=2,padding='same',
activation='relu'))
human_model.add(MaxPooling2D(pool_size=2))
human_model.add(Conv2D(filters=266, kernel_size=2, padding='same',
activation='relu'))
human_model.add(GlobalAveragePooling2D(data_format='channels_last'))
human_model.add(Dense(500, activation='relu'))
human_model.add(Dense(500, activation='relu'))
human_model.add(Dense(1, activation='sigmoid'))

human_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      4640      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        18496     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 128)       32896     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 128)       0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 266)       136458    
_________________________________________________________________
global_average_pooling2d_1 ( (None, 266)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 500)               133500    
_________________________________________________________________
dense_2 (Dense)              (None, 500)               250500    
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 501       
=================================================================
Total params: 577,199.0
Trainable params: 577,199.0
Non-trainable params: 0.0
_________________________________________________________________
```

In [12]:
```
from keras.optimizers import RMSprop
rmsprop_optimizer=RMSprop(lr=0.0001)
human_model.compile(optimizer=rmsprop_optimizer,
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
```

In [13]:
```
from keras.callbacks import ReduceLROnPlateau,EarlyStopping, ModelCheckpoint
from keras.models import load_model

load_trained_model=True
epochs = 100
early_stopping = EarlyStopping(monitor='val_loss', patience=11)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                              patience=2,  verbose=1)

checkpointer = ModelCheckpoint(filepath='saved_models/human_weights.best.from_
scratch.hdf5',
                               verbose=1, save_best_only=True)

if load_trained_model:
    human_model=load_model("human_or_dog.h5")
else:
    human_model.fit(train_tensors, train_labels,
              validation_split=0.10,
              shuffle=True,
              epochs=epochs, batch_size=100, callbacks=[checkpointer,early_sto
pping], verbose=2)
```

In [46]:
```
human_model.load_weights('saved_models/human_weights.best.from_scratch.hdf5')
```

Uses CNN to detect if human in image

In [14]:
```
# returns "True" if face is detected in image stored at img_path
def face_detector3(img_path):
    imgrs = path_to_tensor(img_path=img_path)
    faces = human_model.predict(imgrs)
    faces = faces[0][0]
    return (faces > 0.5)
```

Tests for human images in the following datasets:

1) test set of human images

2) test set of dog images

3) human_files_short dataset

4) dog_files_short dataset

In [15]:
```
human_detection=100*sum(1 for face in human_test_files[:100] if
face_detector3(face))/len(human_test_files[:100])
dog_detection=100*(1-sum(1 for dogface in test_files[:100] if face_detector3(d
ogface))/len(test_files[:100]))

print("Accuracy of human_test_files:",human_detection)
print("Accuracy of (dog) test_files:",dog_detection)

human_detection=100*sum(1 for face in human_files_short if
face_detector3(face))/len(human_files_short)
dog_detection=100*(1-sum(1 for dogface in dog_files_short if face_detector3(do
gface))/len(dog_files_short))

print("Accuracy of human_files_short:",human_detection)
print("Accuracy of dog_files_short:",dog_detection)
```

```
Accuracy of human_test_files: 100.0
Accuracy of (dog) test_files: 100.0
Accuracy of human_files_short: 100.0
Accuracy of dog_files_short: 98.0
```

In [402]:
```
#human_model.save('human_or_dog.h5')
```

Step 2: Detect Dogs

In this section, we use a pre-trained ResNet-50 (http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006) model to detect dogs in images. Our first line of code downloads the ResNet-50 model, along with weights that have been trained on ImageNet (http://www.image-net.org/), a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a). Given an image, this pre-trained ResNet-50 model returns a prediction (derived from the available categories in ImageNet) for the object that is contained in the image.

In [16]:
```
from keras.applications.resnet50 import ResNet50

# define ResNet50 model
ResNet50_model = ResNet50(weights='imagenet')
```

Pre-process the Data

When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape

$$(\text{nb\_samples}, \text{rows}, \text{columns}, \text{channels}),$$

where `nb_samples` corresponds to the total number of images (or samples), and `rows`, `columns`, and `channels` correspond to the number of rows, columns, and channels for each image, respectively.

The `path_to_tensor` function below takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224×224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since we are working with color images, each image has three channels. Likewise, since we are processing a single image (or sample), the returned tensor will always have shape

$$(1, 224, 224, 3).$$

The `paths_to_tensor` function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape

$$(\text{nb\_samples}, 224, 224, 3).$$

Here, `nb_samples` is the number of samples, or number of images, in the supplied array of image paths. It is best to think of `nb_samples` as the number of 3D tensors (where each 3D tensor corresponds to a different image) in your dataset!

In [17]:
```
from keras.preprocessing import image
from tqdm import tqdm

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D
 tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in
tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

Making Predictions with ResNet-50

Getting the 4D tensor ready for ResNet-50, and for any other pre-trained model in Keras, requires some additional processing. First, the RGB image is converted to BGR by reordering the channels. All pre-trained models have the additional normalization step that the mean pixel (expressed in RGB as $[103.939, 116.779, 123.68]$ and calculated from all pixels in all images in ImageNet) must be subtracted from every pixel in each image. This is implemented in the imported function `preprocess_input`. If you're curious, you can check the code for `preprocess_input` here (https://github.com/fchollet/keras/blob/master/keras/applications/imagenet_utils.py).

Now that we have a way to format our image for supplying to ResNet-50, we are now ready to use the model to extract the predictions. This is accomplished with the `predict` method, which returns an array whose i-th entry is the model's predicted probability that the image belongs to the i-th ImageNet category. This is implemented in the `ResNet50_predict_labels` function below.

By taking the argmax of the predicted probability vector, we obtain an integer corresponding to the model's predicted object class, which we can identify with an object category through the use of this dictionary (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a).

In [18]:
```
from keras.applications.resnet50 import preprocess_input, decode_predictions

def ResNet50_predict_labels(img_path):
    # returns prediction vector for image located at img_path
    img = preprocess_input(path_to_tensor(img_path))
    return np.argmax(ResNet50_model.predict(img))
```

Write a Dog Detector

While looking at the dictionary (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a), you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from `'Chihuahua'` to `'Mexican hairless'`. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained ResNet-50 model, we need only check if the `ResNet50_predict_labels` function above returns a value between 151 and 268 (inclusive).

We use these ideas to complete the `dog_detector` function below, which returns `True` if a dog is detected in an image (and `False` if not).

In [19]:
```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    prediction = ResNet50_predict_labels(img_path)
    return ((prediction <= 268) & (prediction >= 151))
```

(IMPLEMENTATION) Assess the Dog Detector

Question 3: Use the code cell below to test the performance of your `dog_detector` function.

- What percentage of the images in `human_files_short` have a detected dog?
- What percentage of the images in `dog_files_short` have a detected dog?

Answer:

1% have detected a dog in human images (99% accuracy)

100% have detected a dog in dog images

In [20]:
```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.

human_detection=100.*(1-sum(1 for face in human_files_short if dog_detector(fa
ce))/len(human_files_short))
dog_detection=100.*(sum(1 for dogface in dog_files_short if dog_detector(dogfa
ce))/len(dog_files_short))
print("Accuracy of humans not detected in dog images:",(human_detection))
print("Accuracy of dogs detected in dog images:",(dog_detection))
```

```
Accuracy of humans not detected in dog images: 99.0
Accuracy of dogs detected in dog images: 100.0
```

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

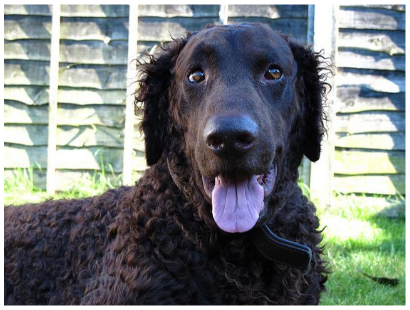
Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. In this step, you will create a CNN that classifies dog breeds. You must create your CNN *from scratch* (so, you can't use transfer learning *yet*!), and you must attain a test accuracy of at least 1%. In Step 5 of this notebook, you will have the opportunity to use transfer learning to create a CNN that attains greatly improved accuracy.

Be careful with adding too many trainable layers! More parameters means longer training, which means you are more likely to need a GPU to accelerate the training process. Thankfully, Keras provides a handy estimate of the time that each epoch is likely to take; you can extrapolate this estimate to figure out how long it will take for your algorithm to train.

We mention that the task of assigning breed to dogs from images is considered exceptionally challenging. To see why, consider that *even a human* would have great difficulty in distinguishing between a Brittany and a Welsh Springer Spaniel.

Brittany	Welsh Springer Spaniel

It is not difficult to find other dog breed pairs with minimal inter-class variation (for instance, Curly-Coated Retrievers and American Water Spaniels).

Curly-Coated Retriever	American Water Spaniel
	

Likewise, recall that labradors come in yellow, chocolate, and black. Your vision-based algorithm will have to conquer this high intra-class variation to determine how to classify all of these different shades as the same breed.

Yellow Labrador	Chocolate Labrador	Black Labrador

We also mention that random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imabalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Remember that the practice is far ahead of the theory in deep learning. Experiment with many different architectures, and trust your intuition. And, of course, have fun!

Pre-process the Data

We rescale the images by dividing every pixel in every image by 255.

In [21]:
```
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

# pre-process the data for Keras
train_tensors = paths_to_tensor(train_files).astype('float32')/255
valid_tensors = paths_to_tensor(valid_files).astype('float32')/255
test_tensors = paths_to_tensor(test_files).astype('float32')/255
```

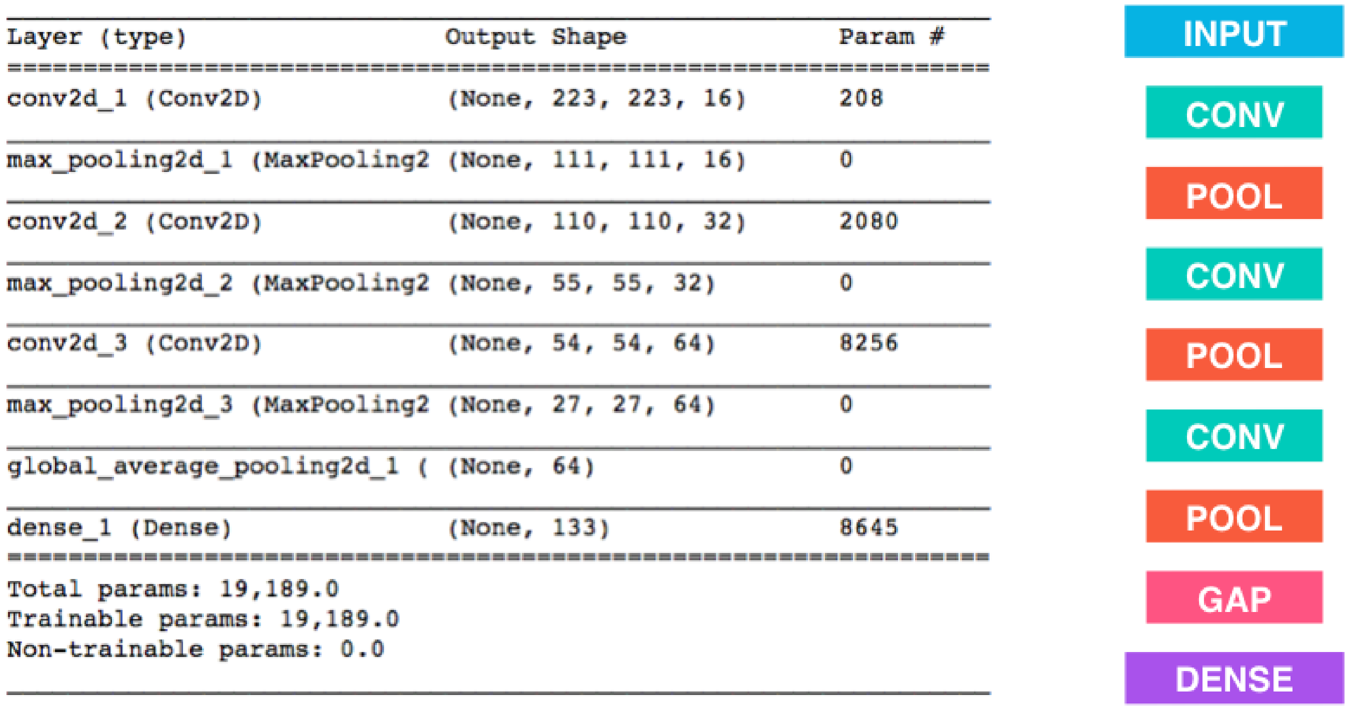
```
100%|████████████████████████████████████
███████████| 6680/6680 [01:02<00:00, 107.30it/s]
100%|████████████████████████████████████
████████████| 835/835 [00:07<00:00, 107.05it/s]
100%|████████████████████████████████████
████████████| 836/836 [00:07<00:00, 117.61it/s]
```

(IMPLEMENTATION) Model Architecture

Create a CNN to classify dog breed. At the end of your code cell block, summarize the layers of your model by executing the line:

```
    model.summary()
```

We have imported some Python modules to get you started, but feel free to import as many modules as you need. If you end up getting stuck, here's a hint that specifies a model that trains relatively fast on CPU and attains >1% test accuracy in 5 epochs:



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 223, 223, 16)      208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 32)      2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 54, 54, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 27, 27, 64)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               8645
=================================================================
Total params: 19,189.0
Trainable params: 19,189.0
Non-trainable params: 0.0
_________________________________________________________________
```

Question 4: Outline the steps you took to get to your final CNN architecture and your reasoning at each step. If you chose to use the hinted architecture above, describe why you think that CNN architecture should work well for the image classification task.

Answer:

There were several iterations to create a decent architecture. Initially I started with a basic CNN, no batchnormalization, dropout, and only a few Conv2d layers. I know that deeper architectures can handle more complex images, so I started increasing the depth of the network. Increasing the depth of the network increases the likelyhood of getting a vanishing gradient.

To compensate for vanishing gradient, I implemeneted BatchNormalization, which normalizes the gradients so that very small numbers are boosted, and very large numbers are reduced. Batch normalization, with Keras should be aligned to the axis with the channls, in this case axis=3, and applied to all layers.

Larger networks also hava a tendency to memorize the dataset as opposed to a generalized smooth prediction. To compensate for this, Dropout was applied, and was most effective on the final fully connected portion of the CNN.

Additionally, the training uses both augmented images as well as raw images, in a hybrid training process of looping the training over augmented images, then raw images. This was very effective for increasing the accuracy.

Further implementations, not implemented, of a better CNN would be to utilize the keras api , not sequential, network model. This would allow for creating more advanced architectures, including a full Resnet50 type model. An approach that I beleive would be very accurate would be to create a mini-Resnet50 type architecture, including shortcut levels, but not having as many levels as the full-blown Resnet50 model.

In [40]:
```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, BatchNo
rmalization, Activation
from keras.layers import Dropout, Flatten, Dense,Input
from keras.models import Sequential

# input_img = Input(shape=(224, 224, 3)) # used for keras api model

### TODO: Define your architecture.
scratch_cnn_model = Sequential()
scratch_cnn_model.add(Conv2D(kernel_initializer='he_normal',
                 input_shape=(224, 224, 3),
                 filters=16, kernel_size=2, padding='same', activation=None))
scratch_cnn_model.add(BatchNormalization(axis=3))
scratch_cnn_model.add(Activation('relu'))
scratch_cnn_model.add(MaxPooling2D(pool_size=2,strides=(1,1)))
scratch_cnn_model.add(Conv2D(kernel_initializer='he_normal',
                 filters=32, kernel_size=2, padding='same', activation=None))
scratch_cnn_model.add(BatchNormalization(axis=3))
scratch_cnn_model.add(Activation('relu'))
scratch_cnn_model.add(MaxPooling2D(pool_size=2,strides=(1,1)))
scratch_cnn_model.add(Conv2D(kernel_initializer='he_normal',
                 filters=64, kernel_size=3,strides=(2,2), padding='same', acti
vation=None))
scratch_cnn_model.add(BatchNormalization(axis=3))
scratch_cnn_model.add(Activation('relu'))
scratch_cnn_model.add(MaxPooling2D(pool_size=3,strides=(2,2)))
scratch_cnn_model.add(Conv2D(kernel_initializer='he_normal',
                 filters=133, kernel_size=3,strides=(2,2), padding='same', act
ivation=None))
scratch_cnn_model.add(BatchNormalization(axis=3))
scratch_cnn_model.add(Activation('relu'))
scratch_cnn_model.add(Conv2D(kernel_initializer='he_normal',
                 filters=133, kernel_size=3,strides=(2,2), padding='same', act
ivation='relu'),)
scratch_cnn_model.add(BatchNormalization(axis=3))
scratch_cnn_model.add(Activation('relu'))
scratch_cnn_model.add(GlobalAveragePooling2D(data_format='channels_last'))
scratch_cnn_model.add(Dense(768, activation='relu'))
scratch_cnn_model.add(Dropout(0.6))
scratch_cnn_model.add(Dense(133, activation='softmax'))
scratch_cnn_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_11 (Conv2D)           (None, 224, 224, 16)      208       
_________________________________________________________________
batch_normalization_6 (Batch (None, 224, 224, 16)      64        
_________________________________________________________________
activation_349 (Activation)  (None, 224, 224, 16)      0         
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 223, 223, 16)      0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 223, 223, 32)      2080      
_________________________________________________________________
batch_normalization_7 (Batch (None, 223, 223, 32)      128       
_________________________________________________________________
activation_350 (Activation)  (None, 223, 223, 32)      0         
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 222, 222, 32)      0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 111, 111, 64)      18496     
_________________________________________________________________
batch_normalization_8 (Batch (None, 111, 111, 64)      256       
_________________________________________________________________
activation_351 (Activation)  (None, 111, 111, 64)      0         
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 55, 55, 64)        0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 28, 28, 133)       76741     
_________________________________________________________________
batch_normalization_9 (Batch (None, 28, 28, 133)       532       
_________________________________________________________________
activation_352 (Activation)  (None, 28, 28, 133)       0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 14, 14, 133)       159334    
_________________________________________________________________
batch_normalization_10 (Batc (None, 14, 14, 133)       532       
_________________________________________________________________
activation_353 (Activation)  (None, 14, 14, 133)       0         
_________________________________________________________________
global_average_pooling2d_4 ( (None, 133)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 768)               102912    
_________________________________________________________________
dropout_5 (Dropout)          (None, 768)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 133)               102277    
=================================================================
Total params: 463,560.0
Trainable params: 462,804.0
Non-trainable params: 756.0
_________________________________________________________________
```

Alternate Scratch CNN Model Attempts

In [ ]:
```
# 41.2 % , augmented hybrid
#
# model = Sequential()
# input_img = Input(shape=(224, 224, 3))
# ### TODO: Define your architecture.
# model = Sequential()
# model.add(Conv2D(kernel_initializer='he_normal',
#                  input_shape=(224, 224, 3),
#                  filters=16, kernel_size=2, padding='same', activation=Non
e))
# model.add(BatchNormalization(axis=3))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=2,strides=(1,1)))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  filters=32, kernel_size=2, padding='same', activation=Non
e))
# model.add(BatchNormalization(axis=3))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=2,strides=(1,1)))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  filters=64, kernel_size=3,strides=(2,2), padding='same', ac
tivation=None))
# model.add(BatchNormalization(axis=3))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=3,strides=(2,2)))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  filters=133, kernel_size=3,strides=(2,2), padding='same', a
ctivation=None))
# model.add(BatchNormalization(axis=3))
# model.add(Activation('relu'))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  filters=133, kernel_size=3,strides=(2,2), padding='same', a
ctivation='relu'),)
# model.add(BatchNormalization(axis=3))
# model.add(Activation('relu'))
# model.add(GlobalAveragePooling2D(data_format='channels_last'))
# model.add(Dense(768, activation='relu'))
# model.add(Dropout(0.6))
# model.add(Dense(133, activation='softmax'))
# model.summary()


# 17 %
# model = Sequential()
# model.add(Dropout(0.1,input_shape=(224, 224, 3)))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=64, kernel_size=2, padding='same', activation=Non
e))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Dropout(0.1))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=128, kernel_size=2, padding='same', activation=Non
e))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Dropout(0.2))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=128, kernel_size=2, padding='same', activation=Non
e))
# # model.add(BatchNormalization(axis=1))
# model.add(Activation('relu'))
# model.add(Dropout(0.2))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=256, kernel_size=2, padding='same', activation=Non
e))
# # model.add(BatchNormalization(axis=1))
# model.add(Activation('relu'))
# model.add(Dropout(0.2))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=500, kernel_size=2, padding='same', activation='rel
u'))
# model.add(Activation('relu'))
# model.add(Dropout(0.3))
# model.add(GlobalAveragePooling2D(data_format='channels_last'))
# model.add(Dense(133, activation='softmax'))
# model.summary()


# 23% , 26.6% w/augmentation
# model = Sequential()
# ### TODO: Define your architecture.
# model = Sequential()
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=64, kernel_size=2, padding='same', activation=None,
input_shape=(224, 224, 3)))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=128, kernel_size=2, padding='same', activation=Non
e))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=128, kernel_size=2, padding='same', activation=Non
e))
# model.add(Activation('relu'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                  filters=256, kernel_size=2, padding='same', activation=Non
e))
# model.add(Activation('relu'))
# model.add(Conv2D(kernel_initializer='he_normal',
#                  bias_initializer='TruncatedNormal',
#                    filters=500, kernel_size=2, padding='same', activation='rel
u'))
# model.add(Activation('relu'))
# model.add(GlobalAveragePooling2D(data_format='channels_last'))
# model.add(Dense(133, activation='softmax'))
# model.summary()



# 20.993 % 33% w/augmentation
### TODO: Define your architecture.
# model = Sequential()
# model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='rel
u',input_shape=(224, 224, 3)))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='rel
u'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='rel
u'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=133, kernel_size=2, padding='same', activation='rel
u'))
# model.add(BatchNormalization(axis=1))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=266, kernel_size=2, padding='same', activation='rel
u'))
# model.add(GlobalAveragePooling2D(data_format='channels_last'))
# model.add(Dense(266, activation='relu'))
# model.add(Dropout(0.2))
# model.add(Dense(133, activation='softmax'))



# 12.32 % (25 epochs)
## TODO: Define your architecture.
# model = Sequential()
# model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='rel
u', input_shape=(224, 224, 3)))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='rel
u'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='rel
u'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=133, kernel_size=2, padding='same', activation='rel
u'))
# model.add(BatchNormalization(axis=1))
# model.add(MaxPooling2D(pool_size=2))
# model.add(GlobalAveragePooling2D(data_format='channels_last'))
# model.add(Dense(266, activation='relu'))
# model.add(Dense(133, activation='softmax'))


# 7.2967% (30 epochs)
### TODO: Define your architecture.
```

```
# model = Sequential()
# model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='rel
u', input_shape=(224, 224, 3)))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='rel
u'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='rel
u'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=133, kernel_size=2, padding='same', activation='rel
u'))
# model.add(BatchNormalization(axis=1))
# model.add(MaxPooling2D(pool_size=2))
# model.add(GlobalAveragePooling2D(data_format='channels_last'))
# model.add(Activation('softmax'))


# 4.7%
### TODO: Define your architecture.
# model = Sequential()
# model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='rel
u', input_shape=(224, 224, 3)))
# model.add(MaxPooling2D(pool_size=2))
# model.add(BatchNormalization(axis=1))
# model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='rel
u'))
# model.add(BatchNormalization(axis=1))
# model.add(MaxPooling2D(pool_size=2))
# model.add(Conv2D(filters=256, kernel_size=2, padding='same', activation='rel
u'))
# model.add(MaxPooling2D(pool_size=2))
# model.add(GlobalAveragePooling2D(data_format='channels_last'))
# # model.add(Flatten())
# model.add(Dense(300, activation='relu'))
# model.add(Dense(200, activation='relu'))
# model.add(Dense(133, activation='softmax'))
```

Compile the Model

In [41]:
```
# model.compile(optimizer=RMSprop(lr=.01), loss='categorical_crossentropy', me
trics=['accuracy'])
scratch_cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])
```

(IMPLEMENTATION) Train the Model

Train your model in the code cell below. Use model checkpointing to save the model that attains the best validation loss.

You are welcome to augment the training data (https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html), but this is not a requirement.

Augmented Data

In [24]:
```
from keras.preprocessing.image import ImageDataGenerator

# create and configure augmented image generator
datagen_train = ImageDataGenerator(
    shear_range=0.2, # randomly shear images  (2%)
    zoom_range=0.08, # randomly zoom images (8%)
    rotation_range=10, # randomly rotate images (10 degree range)
    width_shift_range=0.1,  # randomly shift images horizontally (10% of total
 width)
    height_shift_range=0.1,  # randomly shift images vertically (10% of total
 height)
    horizontal_flip=True) # randomly flip images horizontally

# do not generate augmented images for validation set
datagen_valid = ImageDataGenerator()

# fit augmented image generator on data
datagen_train.fit(train_tensors)
datagen_valid.fit(valid_tensors)
```

Hybrid Augmented / Raw training

This training alternates between training with augmented images and then raw images The training is done for a number of epochs on augmented training(augment_epochs) and trained for a number of epochs on the raw image data(non_augmented_epochs)

In [42]:
```
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.models import load_model
from keras_tqdm import TQDMNotebookCallback


load_cnn=True
train_cnn=False
use_augmentation=True

if load_cnn:
    load_model("mycnn_augmented3.h5")

if train_cnn:
    master_epochs = 200
    augment_epochs=10
    non_augment_epochs=5
    early_stopping = EarlyStopping(monitor='val_loss', patience=3)
    batch_size=100
    checkpointer = ModelCheckpoint(monitor='val_acc',filepath='saved_models/we
ights.best.from_scratch.hdf5',
                               verbose=1, save_best_only=True)
    print(train_tensors.shape[0])
    total_epochs=master_epochs//(augment_epochs+non_augment_epochs)
    for i in range(total_epochs):
        print("Step:",i,"of ",total_epochs)
        if use_augmentation:
            scratch_cnn_model.fit_generator(datagen_train.flow(train_tensors,
train_targets, batch_size=batch_size),
                                steps_per_epoch=train_tensors.shape[0] // batc
h_size,
                                epochs=augment_epochs, verbose=0,
                                validation_data=datagen_valid.flow(valid_tenso
rs, valid_targets, batch_size=batch_size),
                                validation_steps=valid_tensors.shape[0] // bat
ch_size,
                                callbacks=[early_stopping,
                                           checkpointer,
                                           TQDMNotebookCallback(leave_inner=Tr
ue,leave_outer=True)]
                               )
            scratch_cnn_model.fit(train_tensors, train_targets,
                      validation_data=(valid_tensors, valid_targets),
                      epochs=non_augment_epochs, batch_size=batch_size,
                      callbacks=[early_stopping,
                                 checkpointer,
                                 TQDMNotebookCallback(leave_inner=True,leave_o
uter=True)],
                      verbose=0)
```

In [44]:
```
scratch_cnn_model.load_weights('saved_models/weights.best.from_scratch.hdf5')
```

In [45]:
```
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(scratch_cnn_model.predict(np.expand_dims(te
nsor, axis=0))) for tensor in test_tensors]

# report test accuracy
test_accuracy = 100*np.sum(np.array(dog_breed_predictions)==np.argmax(test_tar
gets, axis=1))/len(dog_breed_predictions)
print('CNN from scratch, Test accuracy: %.4f%%' % test_accuracy)
```

```
CNN from scratch, Test accuracy: 41.2679%
```

save model for future runs

In [27]:
```
#scratch_cnn_model.save("mycnn_augmented3.h5")
```

for loading scratch cnn trained only with raw images

In [70]:
```
load_cnn=True
if load_cnn:
    scratch_cnn_model=load_model("mycnn.h5")
```

For Scratch CNN Raw image Training (non-Augmented images), for comparison

In [69]:
```
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.models import load_model

# load_cnn=True
# train_cnn=True

# if load_cnn:
#     scratch_cnn_model=load_model("mycnn.h5")

# if train_cnn:
#     epochs = 100
#     early_stopping = EarlyStopping(monitor='val_loss', patience=11)


#     checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.from_
scratch.hdf5',
#                                verbose=1, save_best_only=True)

#     scratch_cnn_model.fit(train_tensors, train_targets,
#               validation_data=(valid_tensors, valid_targets),
#               epochs=epochs, batch_size=64, callbacks=[checkpointer,early_st
opping], verbose=2)

### TODO: specify the number of epochs that you would like to use to train the
 model.

epochs = 100

### Do NOT modify the code below this line.
scratch_cnn_model.fit(train_tensors, train_targets,
          validation_data=(valid_tensors, valid_targets),
          epochs=epochs, batch_size=20, callbacks=[checkpointer], verbose=1)
```

```
Train on 6680 samples, validate on 835 samples
Epoch 1/100
1280/6680 [====>.........................] - ETA: 357s - loss: 1.4301 - acc:
 0.600 - ETA: 191s - loss: 1.6308 - acc: 0.525 - ETA: 135s - loss: 1.5693 - a
cc: 0.533 - ETA: 107s - loss: 1.5714 - acc: 0.537 - ETA: 90s - loss: 1.6949 -
acc: 0.500 - ETA: 79s - loss: 1.8997 - acc: 0.47 - ETA: 71s - loss: 1.8427 -
 acc: 0.47 - ETA: 65s - loss: 1.7910 - acc: 0.50 - ETA: 60s - loss: 1.9161 -
 acc: 0.48 - ETA: 56s - loss: 1.8926 - acc: 0.49 - ETA: 53s - loss: 1.8894 -
 acc: 0.48 - ETA: 50s - loss: 1.8556 - acc: 0.48 - ETA: 48s - loss: 1.8050 -
 acc: 0.50 - ETA: 46s - loss: 1.7846 - acc: 0.50 - ETA: 44s - loss: 1.7825 -
 acc: 0.51 - ETA: 43s - loss: 1.7870 - acc: 0.51 - ETA: 42s - loss: 1.7442 -
 acc: 0.51 - ETA: 41s - loss: 1.7500 - acc: 0.51 - ETA: 39s - loss: 1.7357 -
 acc: 0.51 - ETA: 38s - loss: 1.7267 - acc: 0.51 - ETA: 38s - loss: 1.7188 -
 acc: 0.51 - ETA: 37s - loss: 1.7462 - acc: 0.50 - ETA: 36s - loss: 1.7467 -
 acc: 0.50 - ETA: 35s - loss: 1.7397 - acc: 0.50 - ETA: 35s - loss: 1.7324 -
 acc: 0.50 - ETA: 34s - loss: 1.7098 - acc: 0.50 - ETA: 34s - loss: 1.7175 -
 acc: 0.50 - ETA: 33s - loss: 1.7273 - acc: 0.50 - ETA: 32s - loss: 1.7248 -
 acc: 0.50 - ETA: 32s - loss: 1.7149 - acc: 0.50 - ETA: 32s - loss: 1.7091 -
 acc: 0.51 - ETA: 31s - loss: 1.7174 - acc: 0.51 - ETA: 31s - loss: 1.7020 -
 acc: 0.51 - ETA: 30s - loss: 1.6933 - acc: 0.51 - ETA: 30s - loss: 1.6985 -
 acc: 0.51 - ETA: 30s - loss: 1.6941 - acc: 0.51 - ETA: 29s - loss: 1.7034 -
 acc: 0.50 - ETA: 29s - loss: 1.7313 - acc: 0.50 - ETA: 29s - loss: 1.7468 -
 acc: 0.49 - ETA: 28s - loss: 1.7553 - acc: 0.49 - ETA: 28s - loss: 1.7525 -
 acc: 0.49 - ETA: 28s - loss: 1.7469 - acc: 0.50 - ETA: 28s - loss: 1.7427 -
 acc: 0.50 - ETA: 27s - loss: 1.7432 - acc: 0.50 - ETA: 27s - loss: 1.7454 -
 acc: 0.50 - ETA: 27s - loss: 1.7338 - acc: 0.50 - ETA: 27s - loss: 1.7264 -
 acc: 0.50 - ETA: 26s - loss: 1.7226 - acc: 0.50 - ETA: 26s - loss: 1.7184 -
 acc: 0.51 - ETA: 26s - loss: 1.7175 - acc: 0.51 - ETA: 26s - loss: 1.7288 -
 acc: 0.50 - ETA: 26s - loss: 1.7198 - acc: 0.51 - ETA: 25s - loss: 1.7147 -
 acc: 0.51 - ETA: 25s - loss: 1.7098 - acc: 0.51 - ETA: 25s - loss: 1.7146 -
 acc: 0.51 - ETA: 25s - loss: 1.7150 - acc: 0.50 - ETA: 25s - loss: 1.7138 -
 acc: 0.50 - ETA: 24s - loss: 1.7135 - acc: 0.51 - ETA: 24s - loss: 1.7100 -
 acc: 0.51 - ETA: 24s - loss: 1.7060 - acc: 0.51 - ETA: 24s - loss: 1.7191 -
 acc: 0.51 - ETA: 24s - loss: 1.7099 - acc: 0.51 - ETA: 24s - loss: 1.7061 -
 acc: 0.51 - ETA: 23s - loss: 1.6983 - acc: 0.5156
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-69-52623ffd67ec> in <module>()
     29 scratch_cnn_model.fit(train_tensors, train_targets, 
     30           validation_data=(valid_tensors, valid_targets),
---> 31           epochs=epochs, batch_size=20, callbacks=[checkpointer], ver
bose=1)

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\keras
\models.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validat
ion_split, validation_data, shuffle, class_weight, sample_weight, initial_epo
ch, **kwargs)
    843                               class_weight=class_weight,
    844                               sample_weight=sample_weight,
--> 845                               initial_epoch=initial_epoch)
    846 
    847     def evaluate(self, x, y, batch_size=32, verbose=1,

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\keras
\engine\training.py in fit(self, x, y, batch_size, epochs, verbose, callback
s, validation_split, validation_data, shuffle, class_weight, sample_weight, i
nitial_epoch, **kwargs)
   1483                               val_f=val_f, val_ins=val_ins, shuffle=s
huffle,
   1484                               callback_metrics=callback_metrics,
-> 1485                               initial_epoch=initial_epoch)
   1486 
   1487     def evaluate(self, x, y, batch_size=32, verbose=1,
sample_weight=None):

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\keras
\engine\training.py in _fit_loop(self, f, ins, out_labels, batch_size, epoch
s, verbose, callbacks, val_f, val_ins, shuffle, callback_metrics, initial_epo
ch)
   1138                 batch_logs['size'] = len(batch_ids)
   1139                 callbacks.on_batch_begin(batch_index, batch_logs)
-> 1140                 outs = f(ins_batch)
   1141                 if not isinstance(outs, list):
   1142                     outs = [outs]

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\keras
\backend\tensorflow_backend.py in __call__(self, inputs)
   2071         session = get_session()
   2072         updated = session.run(self.outputs + [self.updates_op],
-> 2073                               feed_dict=feed_dict)
   2074         return updated[:len(self.outputs)]
   2075 

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\tenso
rflow\python\client\session.py in run(self, fetches, feed_dict, options, run_
metadata)
    765     try:
    766       result = self._run(None, fetches, feed_dict, options_ptr,
--> 767                          run_metadata_ptr)
    768       if run_metadata:
    769         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)
```

```
C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\tenso
rflow\python\client\session.py in _run(self, handle, fetches, feed_dict, opti
ons, run_metadata)
    963     if final_fetches or final_targets:
    964       results = self._do_run(handle, final_targets, final_fetches,
--> 965                              feed_dict_string, options, run_metadata)
    966     else:
    967       results = []

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\tenso
rflow\python\client\session.py in _do_run(self, handle, target_list, fetch_li
st, feed_dict, options, run_metadata)
   1013     if handle is None:
   1014       return self._do_call(_run_fn, self._session, feed_dict, fetch_l
ist,
-> 1015                            target_list, options, run_metadata)
   1016     else:
   1017       return self._do_call(_prun_fn, self._session, handle, feed_dic
t,

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\tenso
rflow\python\client\session.py in _do_call(self, fn, *args)
   1020   def _do_call(self, fn, *args):
   1021     try:
-> 1022       return fn(*args)
   1023     except errors.OpError as e:
   1024       message = compat.as_text(e.message)

C:\Program Files\Anaconda3\envs\tensorflowgpu_w_keras\lib\site-packages\tenso
rflow\python\client\session.py in _run_fn(session, feed_dict, fetch_list, tar
get_list, options, run_metadata)
   1002         return tf_session.TF_Run(session, options,
   1003                                  feed_dict, fetch_list, target_list,
-> 1004                                  status, run_metadata)
   1005 
   1006     def _prun_fn(session, handle, feed_dict, fetch_list):

KeyboardInterrupt:
```

Load the Model with the Best Validation Loss

In [113]:
```
scratch_cnn_model.load_weights('saved_models/weights.best.from_scratch.hdf5')
```

Test the Model

Try out your model on the test dataset of dog images. Ensure that your test accuracy is greater than 1%.

In [72]:
```
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(scratch_cnn_model.predict(np.expand_dims(te
nsor, axis=0))) for tensor in test_tensors]

# report test accuracy
test_accuracy = 100*np.sum(np.array(dog_breed_predictions)==np.argmax(test_tar
gets, axis=1))/len(dog_breed_predictions)
print('Scratch CNN trained w/o augmentation, Test accuracy: %.4f%%' % test_acc
uracy)
```

```
Scratch CNN trained w/o augmentation, Test accuracy: 22.0096%
```

In [132]:
```
model.save("mycnn.h5")
```

Step 4: Use a CNN to Classify Dog Breeds

To reduce training time without sacrificing accuracy, we show you how to train a CNN using transfer learning. In the following step, you will get a chance to use transfer learning to train your own CNN.

Obtain Bottleneck Features

In [73]:
```
bottleneck_features = np.load('bottleneck_features/DogVGG16Data.npz')
train_VGG16 = bottleneck_features['train']
valid_VGG16 = bottleneck_features['valid']
test_VGG16 = bottleneck_features['test']
```

Model Architecture

The model uses the the pre-trained VGG-16 model as a fixed feature extractor, where the last convolutional output of VGG-16 is fed as input to our model. We only add a global average pooling layer and a fully connected layer, where the latter contains one node for each dog category and is equipped with a softmax.

In [74]:
```
VGG16_model = Sequential()
VGG16_model.add(GlobalAveragePooling2D(input_shape=train_VGG16.shape[1:]))
VGG16_model.add(Dense(133, activation='softmax'))

VGG16_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
global_average_pooling2d_6 ( (None, 512)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 133)               68229     
=================================================================
Total params: 68,229.0
Trainable params: 68,229.0
Non-trainable params: 0.0
_________________________________________________________________
```

Compile the Model

In [76]:
```
VGG16_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metr
ics=['accuracy'])
```

Train the Model

In [77]:
```
checkpointer = 
ModelCheckpoint(filepath='saved_models/weights.best.VGG16.hdf5', 
                               verbose=1, save_best_only=True)

VGG16_model.fit(train_VGG16, train_targets, 
          validation_data=(valid_VGG16, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)
```

```
Train on 6680 samples, validate on 835 samples
Epoch 1/20
6620/6680 [============================>.] - ETA: 384s - loss: 14.2269 - acc:
0.0000e+ - ETA: 49s - loss: 14.4546 - acc: 0.0250     - ETA: 28s - loss: 14.8
133 - acc: 0.014 - ETA: 20s - loss: 14.6207 - acc: 0.022 - ETA: 15s - loss: 1
4.4424 - acc: 0.027 - ETA: 12s - loss: 14.3172 - acc: 0.035 - ETA: 10s - los
s: 14.3209 - acc: 0.035 - ETA: 9s - loss: 14.1649 - acc: 0.041 - ETA: 8s - lo
ss: 14.1185 - acc: 0.04 - ETA: 7s - loss: 14.0912 - acc: 0.04 - ETA: 6s - los
s: 14.0792 - acc: 0.04 - ETA: 6s - loss: 13.9321 - acc: 0.05 - ETA: 5s - los
s: 13.9618 - acc: 0.04 - ETA: 5s - loss: 13.8729 - acc: 0.05 - ETA: 4s - los
s: 13.8293 - acc: 0.05 - ETA: 4s - loss: 13.7434 - acc: 0.05 - ETA: 4s - los
s: 13.6510 - acc: 0.05 - ETA: 3s - loss: 13.6095 - acc: 0.06 - ETA: 3s - los
s: 13.5787 - acc: 0.06 - ETA: 3s - loss: 13.4862 - acc: 0.06 - ETA: 3s - los
s: 13.4361 - acc: 0.06 - ETA: 3s - loss: 13.3914 - acc: 0.06 - ETA: 2s - los
s: 13.3495 - acc: 0.06 - ETA: 2s - loss: 13.3076 - acc: 0.07 - ETA: 2s - los
s: 13.2872 - acc: 0.07 - ETA: 2s - loss: 13.2351 - acc: 0.07 - ETA: 2s - los
s: 13.2072 - acc: 0.07 - ETA: 2s - loss: 13.1713 - acc: 0.07 - ETA: 2s - los
s: 13.1224 - acc: 0.07 - ETA: 1s - loss: 13.0547 - acc: 0.08 - ETA: 1s - los
s: 13.0211 - acc: 0.08 - ETA: 1s - loss: 12.9760 - acc: 0.08 - ETA: 1s - los
s: 12.9442 - acc: 0.08 - ETA: 1s - loss: 12.9032 - acc: 0.09 - ETA: 1s - los
s: 12.8884 - acc: 0.09 - ETA: 1s - loss: 12.8763 - acc: 0.09 - ETA: 1s - los
s: 12.8313 - acc: 0.09 - ETA: 1s - loss: 12.7963 - acc: 0.10 - ETA: 1s - los
s: 12.7705 - acc: 0.10 - ETA: 0s - loss: 12.7516 - acc: 0.10 - ETA: 0s - los
s: 12.7291 - acc: 0.10 - ETA: 0s - loss: 12.6884 - acc: 0.10 - ETA: 0s - los
s: 12.6438 - acc: 0.10 - ETA: 0s - loss: 12.6191 - acc: 0.11 - ETA: 0s - los
s: 12.5667 - acc: 0.11 - ETA: 0s - loss: 12.5324 - acc: 0.11 - ETA: 0s - los
s: 12.4732 - acc: 0.12 - ETA: 0s - loss: 12.4415 - acc: 0.12 - ETA: 0s - los
s: 12.3956 - acc: 0.12 - ETA: 0s - loss: 12.3910 - acc: 0.12 - ETA: 0s - los
s: 12.3566 - acc: 0.1272Epoch 00000: val_loss improved from inf to 10.93638,
 saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 4s - loss: 12.3261 - acc: 0.1295
- val_loss: 10.9364 - val_acc: 0.2096
Epoch 2/20
6560/6680 [============================>.] - ETA: 2s - loss: 9.0819 - acc: 0.
350 - ETA: 2s - loss: 10.2858 - acc: 0.25 - ETA: 2s - loss: 10.7918 - acc: 0.
24 - ETA: 2s - loss: 10.9475 - acc: 0.23 - ETA: 2s - loss: 10.8840 - acc: 0.2
3 - ETA: 2s - loss: 10.8430 - acc: 0.23 - ETA: 2s - loss: 10.7407 - acc: 0.23
- ETA: 2s - loss: 10.6221 - acc: 0.24 - ETA: 2s - loss: 10.7095 - acc: 0.24 -
ETA: 2s - loss: 10.6602 - acc: 0.24 - ETA: 2s - loss: 10.5683 - acc: 0.25 - E
TA: 2s - loss: 10.5898 - acc: 0.25 - ETA: 2s - loss: 10.6363 - acc: 0.25 - ET
A: 2s - loss: 10.5981 - acc: 0.25 - ETA: 1s - loss: 10.7005 - acc: 0.24 - ET
A: 1s - loss: 10.7129 - acc: 0.24 - ETA: 1s - loss: 10.6935 - acc: 0.24 - ET
A: 1s - loss: 10.6595 - acc: 0.24 - ETA: 1s - loss: 10.6720 - acc: 0.24 - ET
A: 1s - loss: 10.6755 - acc: 0.24 - ETA: 1s - loss: 10.6928 - acc: 0.24 - ET
A: 1s - loss: 10.6776 - acc: 0.25 - ETA: 1s - loss: 10.6777 - acc: 0.25 - ET
A: 1s - loss: 10.6251 - acc: 0.25 - ETA: 1s - loss: 10.6706 - acc: 0.25 - ET
A: 1s - loss: 10.6158 - acc: 0.25 - ETA: 1s - loss: 10.6220 - acc: 0.25 - ET
A: 1s - loss: 10.6027 - acc: 0.25 - ETA: 1s - loss: 10.5921 - acc: 0.25 - ET
A: 1s - loss: 10.5609 - acc: 0.26 - ETA: 1s - loss: 10.5814 - acc: 0.26 - ET
A: 1s - loss: 10.5822 - acc: 0.26 - ETA: 0s - loss: 10.5879 - acc: 0.26 - ET
A: 0s - loss: 10.5681 - acc: 0.26 - ETA: 0s - loss: 10.5769 - acc: 0.26 - ET
A: 0s - loss: 10.5648 - acc: 0.26 - ETA: 0s - loss: 10.5370 - acc: 0.26 - ET
A: 0s - loss: 10.5359 - acc: 0.26 - ETA: 0s - loss: 10.5421 - acc: 0.26 - ET
A: 0s - loss: 10.5458 - acc: 0.27 - ETA: 0s - loss: 10.5469 - acc: 0.26 - ET
A: 0s - loss: 10.5530 - acc: 0.26 - ETA: 0s - loss: 10.5310 - acc: 0.27 - ET
A: 0s - loss: 10.4988 - acc: 0.27 - ETA: 0s - loss: 10.5073 - acc: 0.27 - ET
A: 0s - loss: 10.4829 - acc: 0.27 - ETA: 0s - loss: 10.4670 - acc: 0.27 - ET
```

```
A: 0s - loss: 10.4686 - acc: 0.27 - ETA: 0s - loss: 10.4590 - acc: 0.27 - ET
A: 0s - loss: 10.4280 - acc: 0.2767Epoch 00001: val_loss improved from 10.936
38 to 10.33657, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 10.4254 - acc: 0.2771
- val_loss: 10.3366 - val_acc: 0.2826
Epoch 3/20
6600/6680 [============================>.] - ETA: 2s - loss: 9.3714 - acc: 0.
400 - ETA: 2s - loss: 9.6103 - acc: 0.375 - ETA: 2s - loss: 9.7222 - acc: 0.3
53 - ETA: 2s - loss: 9.6530 - acc: 0.363 - ETA: 2s - loss: 9.9581 - acc: 0.34
3 - ETA: 2s - loss: 10.0748 - acc: 0.33 - ETA: 2s - loss: 10.0800 - acc: 0.32
- ETA: 2s - loss: 10.0435 - acc: 0.32 - ETA: 2s - loss: 9.9488 - acc: 0.3339
 - ETA: 2s - loss: 9.9172 - acc: 0.338 - ETA: 2s - loss: 9.8670 - acc: 0.337
 - ETA: 2s - loss: 9.8587 - acc: 0.337 - ETA: 1s - loss: 9.8400 - acc: 0.336
 - ETA: 1s - loss: 9.8231 - acc: 0.340 - ETA: 1s - loss: 9.8354 - acc: 0.340
 - ETA: 1s - loss: 9.8156 - acc: 0.341 - ETA: 1s - loss: 9.8053 - acc: 0.341
 - ETA: 1s - loss: 9.8180 - acc: 0.340 - ETA: 1s - loss: 9.8255 - acc: 0.340
 - ETA: 1s - loss: 9.7914 - acc: 0.342 - ETA: 1s - loss: 9.8130 - acc: 0.341
 - ETA: 1s - loss: 9.8417 - acc: 0.340 - ETA: 1s - loss: 9.8068 - acc: 0.343
 - ETA: 1s - loss: 9.8454 - acc: 0.340 - ETA: 1s - loss: 9.8622 - acc: 0.339
 - ETA: 1s - loss: 9.8608 - acc: 0.338 - ETA: 1s - loss: 9.8911 - acc: 0.336
 - ETA: 1s - loss: 9.9084 - acc: 0.334 - ETA: 1s - loss: 9.9603 - acc: 0.331
 - ETA: 1s - loss: 9.9440 - acc: 0.332 - ETA: 1s - loss: 9.9512 - acc: 0.331
 - ETA: 0s - loss: 9.9514 - acc: 0.330 - ETA: 0s - loss: 9.9505 - acc: 0.331
 - ETA: 0s - loss: 9.9512 - acc: 0.332 - ETA: 0s - loss: 9.9657 - acc: 0.331
 - ETA: 0s - loss: 9.9752 - acc: 0.331 - ETA: 0s - loss: 9.9583 - acc: 0.332
 - ETA: 0s - loss: 9.9447 - acc: 0.332 - ETA: 0s - loss: 9.9565 - acc: 0.331
 - ETA: 0s - loss: 9.9519 - acc: 0.332 - ETA: 0s - loss: 9.9283 - acc: 0.333
 - ETA: 0s - loss: 9.8981 - acc: 0.335 - ETA: 0s - loss: 9.9132 - acc: 0.334
 - ETA: 0s - loss: 9.9216 - acc: 0.334 - ETA: 0s - loss: 9.9274 - acc: 0.334
 - ETA: 0s - loss: 9.8946 - acc: 0.336 - ETA: 0s - loss: 9.8924 - acc: 0.336
 - ETA: 0s - loss: 9.9153 - acc: 0.335 - ETA: 0s - loss: 9.9298 - acc: 0.3353
Epoch 00002: val_loss improved from 10.33657 to 10.08925, saving model to sav
ed_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 9.9298 - acc: 0.3355
 - val_loss: 10.0893 - val_acc: 0.3198
Epoch 4/20
6640/6680 [============================>.] - ETA: 2s - loss: 8.5993 - acc: 0.
400 - ETA: 2s - loss: 8.5315 - acc: 0.431 - ETA: 2s - loss: 8.7791 - acc: 0.4
21 - ETA: 2s - loss: 8.8842 - acc: 0.422 - ETA: 2s - loss: 9.2011 - acc: 0.40
0 - ETA: 2s - loss: 9.4109 - acc: 0.388 - ETA: 2s - loss: 9.4378 - acc: 0.384
- ETA: 2s - loss: 9.5725 - acc: 0.379 - ETA: 2s - loss: 9.5411 - acc: 0.378 -
ETA: 2s - loss: 9.4992 - acc: 0.383 - ETA: 2s - loss: 9.4305 - acc: 0.385 - E
TA: 1s - loss: 9.4688 - acc: 0.384 - ETA: 1s - loss: 9.5448 - acc: 0.378 - ET
A: 1s - loss: 9.5688 - acc: 0.376 - ETA: 1s - loss: 9.6434 - acc: 0.370 - ET
A: 1s - loss: 9.6342 - acc: 0.370 - ETA: 1s - loss: 9.6210 - acc: 0.371 - ET
A: 1s - loss: 9.6856 - acc: 0.366 - ETA: 1s - loss: 9.7043 - acc: 0.365 - ET
A: 1s - loss: 9.7172 - acc: 0.364 - ETA: 1s - loss: 9.6908 - acc: 0.365 - ET
A: 1s - loss: 9.7029 - acc: 0.365 - ETA: 1s - loss: 9.7467 - acc: 0.363 - ET
A: 1s - loss: 9.7759 - acc: 0.361 - ETA: 1s - loss: 9.8028 - acc: 0.360 - ET
A: 1s - loss: 9.8077 - acc: 0.360 - ETA: 1s - loss: 9.7809 - acc: 0.361 - ET
A: 1s - loss: 9.8093 - acc: 0.360 - ETA: 1s - loss: 9.8579 - acc: 0.357 - ET
A: 1s - loss: 9.8846 - acc: 0.355 - ETA: 0s - loss: 9.8309 - acc: 0.358 - ET
A: 0s - loss: 9.8394 - acc: 0.357 - ETA: 0s - loss: 9.8034 - acc: 0.359 - ET
A: 0s - loss: 9.7815 - acc: 0.361 - ETA: 0s - loss: 9.7766 - acc: 0.361 - ET
A: 0s - loss: 9.7674 - acc: 0.362 - ETA: 0s - loss: 9.7720 - acc: 0.362 - ET
A: 0s - loss: 9.7662 - acc: 0.363 - ETA: 0s - loss: 9.7564 - acc: 0.364 - ET
A: 0s - loss: 9.7716 - acc: 0.363 - ETA: 0s - loss: 9.7658 - acc: 0.363 - ET
```

```
A: 0s - loss: 9.7597 - acc: 0.364 - ETA: 0s - loss: 9.7319 - acc: 0.365 - ET
A: 0s - loss: 9.7515 - acc: 0.364 - ETA: 0s - loss: 9.7307 - acc: 0.365 - ET
A: 0s - loss: 9.7426 - acc: 0.364 - ETA: 0s - loss: 9.7385 - acc: 0.364 - ET
A: 0s - loss: 9.7449 - acc: 0.364 - ETA: 0s - loss: 9.7563 - acc: 0.3640Epoch
00003: val_loss did not improve
6680/6680 [==============================] - 2s - loss: 9.7508 - acc: 0.3641
 - val_loss: 10.1114 - val_acc: 0.3138
Epoch 5/20
6620/6680 [============================>.] - ETA: 2s - loss: 14.5063 - acc:
 0.10 - ETA: 2s - loss: 11.1172 - acc: 0.29 - ETA: 2s - loss: 10.2141 - acc:
 0.34 - ETA: 2s - loss: 10.1867 - acc: 0.34 - ETA: 2s - loss: 10.0405 - acc:
 0.35 - ETA: 2s - loss: 9.9029 - acc: 0.3667 - ETA: 2s - loss: 9.8974 - acc:
 0.366 - ETA: 2s - loss: 9.9277 - acc: 0.365 - ETA: 2s - loss: 9.9346 - acc:
 0.364 - ETA: 2s - loss: 9.9418 - acc: 0.363 - ETA: 2s - loss: 9.9008 - acc:
 0.365 - ETA: 2s - loss: 9.9110 - acc: 0.365 - ETA: 1s - loss: 9.8290 - acc:
 0.371 - ETA: 1s - loss: 9.8482 - acc: 0.369 - ETA: 1s - loss: 9.8787 - acc:
 0.365 - ETA: 1s - loss: 9.8369 - acc: 0.367 - ETA: 1s - loss: 9.8267 - acc:
 0.367 - ETA: 1s - loss: 9.7893 - acc: 0.370 - ETA: 1s - loss: 9.7811 - acc:
 0.370 - ETA: 1s - loss: 9.8251 - acc: 0.366 - ETA: 1s - loss: 9.8342 - acc:
 0.366 - ETA: 1s - loss: 9.8280 - acc: 0.367 - ETA: 1s - loss: 9.8799 - acc:
 0.364 - ETA: 1s - loss: 9.8832 - acc: 0.363 - ETA: 1s - loss: 9.9103 - acc:
 0.361 - ETA: 1s - loss: 9.8913 - acc: 0.362 - ETA: 1s - loss: 9.8904 - acc:
 0.361 - ETA: 1s - loss: 9.9069 - acc: 0.360 - ETA: 1s - loss: 9.9117 - acc:
 0.360 - ETA: 1s - loss: 9.8763 - acc: 0.362 - ETA: 1s - loss: 9.8432 - acc:
 0.364 - ETA: 1s - loss: 9.8311 - acc: 0.364 - ETA: 0s - loss: 9.8116 - acc:
 0.364 - ETA: 0s - loss: 9.7676 - acc: 0.367 - ETA: 0s - loss: 9.7516 - acc:
 0.368 - ETA: 0s - loss: 9.7317 - acc: 0.369 - ETA: 0s - loss: 9.6888 - acc:
 0.371 - ETA: 0s - loss: 9.6800 - acc: 0.372 - ETA: 0s - loss: 9.6675 - acc:
 0.372 - ETA: 0s - loss: 9.6389 - acc: 0.374 - ETA: 0s - loss: 9.6387 - acc:
 0.374 - ETA: 0s - loss: 9.6251 - acc: 0.375 - ETA: 0s - loss: 9.6225 - acc:
 0.375 - ETA: 0s - loss: 9.5875 - acc: 0.377 - ETA: 0s - loss: 9.5926 - acc:
 0.376 - ETA: 0s - loss: 9.5710 - acc: 0.377 - ETA: 0s - loss: 9.5840 - acc:
 0.377 - ETA: 0s - loss: 9.5812 - acc: 0.376 - ETA: 0s - loss: 9.5912 - acc:
 0.376 - ETA: 0s - loss: 9.6016 - acc: 0.3758Epoch 00004: val_loss improved f
rom 10.08925 to 9.79922, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 9.6052 - acc: 0.3756
 - val_loss: 9.7992 - val_acc: 0.3377
Epoch 6/20
6560/6680 [============================>.] - ETA: 2s - loss: 5.8366 - acc: 0.
600 - ETA: 2s - loss: 9.3997 - acc: 0.406 - ETA: 2s - loss: 9.7227 - acc: 0.3
83 - ETA: 2s - loss: 9.6124 - acc: 0.393 - ETA: 2s - loss: 9.6752 - acc: 0.38
3 - ETA: 2s - loss: 9.7473 - acc: 0.382 - ETA: 2s - loss: 9.5782 - acc: 0.389
- ETA: 2s - loss: 9.5571 - acc: 0.388 - ETA: 2s - loss: 9.6079 - acc: 0.384 -
ETA: 2s - loss: 9.7418 - acc: 0.374 - ETA: 2s - loss: 9.6857 - acc: 0.378 - E
TA: 2s - loss: 9.5873 - acc: 0.382 - ETA: 1s - loss: 9.5396 - acc: 0.384 - ET
A: 1s - loss: 9.5314 - acc: 0.383 - ETA: 1s - loss: 9.4725 - acc: 0.386 - ET
A: 1s - loss: 9.3997 - acc: 0.391 - ETA: 1s - loss: 9.4287 - acc: 0.388 - ET
A: 1s - loss: 9.4459 - acc: 0.386 - ETA: 1s - loss: 9.4742 - acc: 0.383 - ET
A: 1s - loss: 9.4571 - acc: 0.384 - ETA: 1s - loss: 9.3788 - acc: 0.387 - ET
A: 1s - loss: 9.3846 - acc: 0.387 - ETA: 1s - loss: 9.3593 - acc: 0.389 - ET
A: 1s - loss: 9.2991 - acc: 0.390 - ETA: 1s - loss: 9.3189 - acc: 0.389 - ET
A: 1s - loss: 9.2760 - acc: 0.392 - ETA: 1s - loss: 9.2857 - acc: 0.392 - ET
A: 1s - loss: 9.3340 - acc: 0.389 - ETA: 1s - loss: 9.3347 - acc: 0.389 - ET
A: 1s - loss: 9.3041 - acc: 0.391 - ETA: 1s - loss: 9.3042 - acc: 0.391 - ET
A: 0s - loss: 9.2713 - acc: 0.393 - ETA: 0s - loss: 9.2801 - acc: 0.392 - ET
A: 0s - loss: 9.3023 - acc: 0.390 - ETA: 0s - loss: 9.2894 - acc: 0.391 - ET
A: 0s - loss: 9.2831 - acc: 0.391 - ETA: 0s - loss: 9.2783 - acc: 0.391 - ET
```

```
A: 0s - loss: 9.2734 - acc: 0.392 - ETA: 0s - loss: 9.2584 - acc: 0.394 - ET
A: 0s - loss: 9.2691 - acc: 0.393 - ETA: 0s - loss: 9.2811 - acc: 0.393 - ET
A: 0s - loss: 9.2542 - acc: 0.395 - ETA: 0s - loss: 9.2586 - acc: 0.394 - ET
A: 0s - loss: 9.2547 - acc: 0.395 - ETA: 0s - loss: 9.2327 - acc: 0.397 - ET
A: 0s - loss: 9.2094 - acc: 0.398 - ETA: 0s - loss: 9.2083 - acc: 0.398 - ET
A: 0s - loss: 9.1890 - acc: 0.400 - ETA: 0s - loss: 9.2037 - acc: 0.3998Epoch
00005: val_loss improved from 9.79922 to 9.49160, saving model to saved_model
s/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 9.1644 - acc: 0.4027
 - val_loss: 9.4916 - val_acc: 0.3521
Epoch 7/20
6660/6680 [============================>.] - ETA: 2s - loss: 8.0602 - acc: 0.
500 - ETA: 2s - loss: 7.6395 - acc: 0.512 - ETA: 2s - loss: 8.8214 - acc: 0.4
40 - ETA: 2s - loss: 9.0367 - acc: 0.421 - ETA: 2s - loss: 8.9754 - acc: 0.42
5 - ETA: 2s - loss: 9.1653 - acc: 0.410 - ETA: 2s - loss: 9.1942 - acc: 0.408
- ETA: 2s - loss: 9.1795 - acc: 0.410 - ETA: 2s - loss: 9.1845 - acc: 0.413 -
ETA: 2s - loss: 9.1785 - acc: 0.415 - ETA: 2s - loss: 9.1984 - acc: 0.413 - E
TA: 2s - loss: 9.2581 - acc: 0.410 - ETA: 2s - loss: 9.1986 - acc: 0.411 - ET
A: 1s - loss: 9.2182 - acc: 0.409 - ETA: 1s - loss: 9.1334 - acc: 0.414 - ET
A: 1s - loss: 9.1243 - acc: 0.415 - ETA: 1s - loss: 9.0964 - acc: 0.417 - ET
A: 1s - loss: 9.1187 - acc: 0.416 - ETA: 1s - loss: 9.0875 - acc: 0.417 - ET
A: 1s - loss: 9.0515 - acc: 0.418 - ETA: 1s - loss: 9.0312 - acc: 0.420 - ET
A: 1s - loss: 9.0458 - acc: 0.420 - ETA: 1s - loss: 9.0711 - acc: 0.418 - ET
A: 1s - loss: 9.0308 - acc: 0.421 - ETA: 1s - loss: 9.0483 - acc: 0.420 - ET
A: 1s - loss: 9.0325 - acc: 0.420 - ETA: 1s - loss: 9.0714 - acc: 0.418 - ET
A: 1s - loss: 9.0675 - acc: 0.418 - ETA: 1s - loss: 9.0501 - acc: 0.419 - ET
A: 1s - loss: 9.0563 - acc: 0.418 - ETA: 1s - loss: 9.0671 - acc: 0.418 - ET
A: 0s - loss: 9.0449 - acc: 0.419 - ETA: 0s - loss: 9.0413 - acc: 0.418 - ET
A: 0s - loss: 9.0245 - acc: 0.419 - ETA: 0s - loss: 9.0211 - acc: 0.420 - ET
A: 0s - loss: 9.0098 - acc: 0.420 - ETA: 0s - loss: 8.9707 - acc: 0.423 - ET
A: 0s - loss: 8.9562 - acc: 0.424 - ETA: 0s - loss: 8.9609 - acc: 0.424 - ET
A: 0s - loss: 8.9581 - acc: 0.424 - ETA: 0s - loss: 8.9454 - acc: 0.425 - ET
A: 0s - loss: 8.9602 - acc: 0.423 - ETA: 0s - loss: 8.9476 - acc: 0.424 - ET
A: 0s - loss: 8.9494 - acc: 0.424 - ETA: 0s - loss: 8.9829 - acc: 0.422 - ET
A: 0s - loss: 8.9946 - acc: 0.422 - ETA: 0s - loss: 8.9949 - acc: 0.422 - ET
A: 0s - loss: 8.9962 - acc: 0.422 - ETA: 0s - loss: 8.9677 - acc: 0.424 - ET
A: 0s - loss: 8.9900 - acc: 0.4227Epoch 00006: val_loss improved from 9.49160
to 9.44241, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.9872 - acc: 0.4229
 - val_loss: 9.4424 - val_acc: 0.3581
Epoch 8/20
6600/6680 [============================>.] - ETA: 2s - loss: 10.4816 - acc:
 0.35 - ETA: 2s - loss: 9.9776 - acc: 0.3813 - ETA: 2s - loss: 8.6026 - acc:
 0.466 - ETA: 2s - loss: 8.6228 - acc: 0.461 - ETA: 2s - loss: 8.5028 - acc:
 0.469 - ETA: 2s - loss: 8.4326 - acc: 0.472 - ETA: 2s - loss: 8.6922 - acc:
 0.453 - ETA: 2s - loss: 8.7206 - acc: 0.447 - ETA: 2s - loss: 8.7647 - acc:
 0.442 - ETA: 2s - loss: 8.6654 - acc: 0.448 - ETA: 2s - loss: 8.6769 - acc:
 0.448 - ETA: 2s - loss: 8.6981 - acc: 0.447 - ETA: 1s - loss: 8.7030 - acc:
 0.446 - ETA: 1s - loss: 8.7179 - acc: 0.445 - ETA: 1s - loss: 8.7868 - acc:
 0.442 - ETA: 1s - loss: 8.7617 - acc: 0.443 - ETA: 1s - loss: 8.7641 - acc:
 0.443 - ETA: 1s - loss: 8.7854 - acc: 0.442 - ETA: 1s - loss: 8.7832 - acc:
 0.441 - ETA: 1s - loss: 8.8006 - acc: 0.440 - ETA: 1s - loss: 8.8188 - acc:
 0.438 - ETA: 1s - loss: 8.8884 - acc: 0.435 - ETA: 1s - loss: 8.8506 - acc:
 0.436 - ETA: 1s - loss: 8.8171 - acc: 0.439 - ETA: 1s - loss: 8.8290 - acc:
 0.439 - ETA: 1s - loss: 8.8404 - acc: 0.438 - ETA: 1s - loss: 8.8322 - acc:
 0.439 - ETA: 1s - loss: 8.8513 - acc: 0.438 - ETA: 1s - loss: 8.8438 - acc:
 0.438 - ETA: 1s - loss: 8.8334 - acc: 0.439 - ETA: 1s - loss: 8.8630 - acc:
```

```
 0.437 - ETA: 0s - loss: 8.8517 - acc: 0.438 - ETA: 0s - loss: 8.8169 - acc:
 0.440 - ETA: 0s - loss: 8.8229 - acc: 0.440 - ETA: 0s - loss: 8.8267 - acc:
 0.439 - ETA: 0s - loss: 8.8467 - acc: 0.438 - ETA: 0s - loss: 8.8454 - acc:
 0.438 - ETA: 0s - loss: 8.8794 - acc: 0.437 - ETA: 0s - loss: 8.8872 - acc:
 0.436 - ETA: 0s - loss: 8.9143 - acc: 0.435 - ETA: 0s - loss: 8.9238 - acc:
 0.433 - ETA: 0s - loss: 8.9236 - acc: 0.433 - ETA: 0s - loss: 8.9320 - acc:
 0.432 - ETA: 0s - loss: 8.9407 - acc: 0.431 - ETA: 0s - loss: 8.9176 - acc:
 0.433 - ETA: 0s - loss: 8.9290 - acc: 0.432 - ETA: 0s - loss: 8.9226 - acc:
 0.433 - ETA: 0s - loss: 8.9353 - acc: 0.432 - ETA: 0s - loss: 8.9447 - acc:
 0.432 - ETA: 0s - loss: 8.9335 - acc: 0.4330Epoch 00007: val_loss improved f
rom 9.44241 to 9.41935, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.9425 - acc: 0.4325
 - val_loss: 9.4193 - val_acc: 0.3605
Epoch 9/20
6640/6680 [============================>.] - ETA: 2s - loss: 11.2829 - acc:
 0.30 - ETA: 2s - loss: 8.6706 - acc: 0.4625 - ETA: 2s - loss: 8.9794 - acc:
 0.443 - ETA: 2s - loss: 8.5831 - acc: 0.465 - ETA: 2s - loss: 8.8273 - acc:
 0.450 - ETA: 2s - loss: 8.7911 - acc: 0.452 - ETA: 2s - loss: 8.7781 - acc:
 0.452 - ETA: 2s - loss: 9.0292 - acc: 0.437 - ETA: 2s - loss: 9.0245 - acc:
 0.437 - ETA: 2s - loss: 9.0377 - acc: 0.434 - ETA: 2s - loss: 9.0230 - acc:
 0.435 - ETA: 1s - loss: 8.9637 - acc: 0.437 - ETA: 1s - loss: 8.9769 - acc:
 0.435 - ETA: 1s - loss: 8.9247 - acc: 0.439 - ETA: 1s - loss: 8.9374 - acc:
 0.439 - ETA: 1s - loss: 9.0398 - acc: 0.433 - ETA: 1s - loss: 9.0537 - acc:
 0.431 - ETA: 1s - loss: 9.0771 - acc: 0.430 - ETA: 1s - loss: 9.1075 - acc:
 0.428 - ETA: 1s - loss: 9.0756 - acc: 0.429 - ETA: 1s - loss: 9.0656 - acc:
 0.430 - ETA: 1s - loss: 9.0856 - acc: 0.428 - ETA: 1s - loss: 9.0858 - acc:
 0.428 - ETA: 1s - loss: 9.0719 - acc: 0.429 - ETA: 1s - loss: 9.0410 - acc:
 0.431 - ETA: 1s - loss: 9.0488 - acc: 0.431 - ETA: 1s - loss: 9.0303 - acc:
 0.432 - ETA: 1s - loss: 9.0245 - acc: 0.433 - ETA: 1s - loss: 9.0033 - acc:
 0.434 - ETA: 1s - loss: 8.9655 - acc: 0.436 - ETA: 1s - loss: 8.9977 - acc:
 0.434 - ETA: 0s - loss: 8.9483 - acc: 0.437 - ETA: 0s - loss: 8.9721 - acc:
 0.436 - ETA: 0s - loss: 8.9533 - acc: 0.437 - ETA: 0s - loss: 8.9713 - acc:
 0.436 - ETA: 0s - loss: 8.9658 - acc: 0.436 - ETA: 0s - loss: 8.9732 - acc:
 0.436 - ETA: 0s - loss: 8.9676 - acc: 0.436 - ETA: 0s - loss: 8.9928 - acc:
 0.435 - ETA: 0s - loss: 8.9752 - acc: 0.436 - ETA: 0s - loss: 8.9349 - acc:
 0.438 - ETA: 0s - loss: 8.9419 - acc: 0.438 - ETA: 0s - loss: 8.9575 - acc:
 0.437 - ETA: 0s - loss: 8.9550 - acc: 0.437 - ETA: 0s - loss: 8.9433 - acc:
 0.437 - ETA: 0s - loss: 8.9353 - acc: 0.438 - ETA: 0s - loss: 8.9171 - acc:
 0.438 - ETA: 0s - loss: 8.9160 - acc: 0.438 - ETA: 0s - loss: 8.9162 - acc:
 0.4384Epoch 00008: val_loss improved from 9.41935 to 9.38226, saving model t
o saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.9183 - acc: 0.4383
 - val_loss: 9.3823 - val_acc: 0.3713
Epoch 10/20
6660/6680 [============================>.] - ETA: 2s - loss: 8.8656 - acc: 0.
450 - ETA: 2s - loss: 9.7340 - acc: 0.381 - ETA: 2s - loss: 9.6053 - acc: 0.3
86 - ETA: 2s - loss: 9.2651 - acc: 0.411 - ETA: 2s - loss: 9.2151 - acc: 0.41
0 - ETA: 2s - loss: 9.4007 - acc: 0.398 - ETA: 2s - loss: 9.3280 - acc: 0.402
- ETA: 2s - loss: 9.2956 - acc: 0.405 - ETA: 2s - loss: 9.2862 - acc: 0.408 -
ETA: 2s - loss: 9.2220 - acc: 0.413 - ETA: 2s - loss: 9.0107 - acc: 0.426 - E
TA: 1s - loss: 8.9863 - acc: 0.427 - ETA: 1s - loss: 8.9893 - acc: 0.426 - ET
A: 1s - loss: 9.0193 - acc: 0.425 - ETA: 1s - loss: 8.9161 - acc: 0.432 - ET
A: 1s - loss: 8.9588 - acc: 0.430 - ETA: 1s - loss: 8.9186 - acc: 0.433 - ET
A: 1s - loss: 8.9669 - acc: 0.431 - ETA: 1s - loss: 8.9185 - acc: 0.433 - ET
A: 1s - loss: 8.8933 - acc: 0.435 - ETA: 1s - loss: 8.9266 - acc: 0.434 - ET
A: 1s - loss: 8.9333 - acc: 0.434 - ETA: 1s - loss: 8.8786 - acc: 0.437 - ET
A: 1s - loss: 8.8534 - acc: 0.439 - ETA: 1s - loss: 8.8443 - acc: 0.440 - ET
```

```
A: 1s - loss: 8.8204 - acc: 0.441 - ETA: 1s - loss: 8.7721 - acc: 0.444 - ET
A: 1s - loss: 8.7513 - acc: 0.446 - ETA: 1s - loss: 8.7663 - acc: 0.445 - ET
A: 1s - loss: 8.7873 - acc: 0.444 - ETA: 0s - loss: 8.7857 - acc: 0.444 - ET
A: 0s - loss: 8.7772 - acc: 0.445 - ETA: 0s - loss: 8.7472 - acc: 0.446 - ET
A: 0s - loss: 8.7578 - acc: 0.446 - ETA: 0s - loss: 8.7584 - acc: 0.446 - ET
A: 0s - loss: 8.7420 - acc: 0.447 - ETA: 0s - loss: 8.7311 - acc: 0.447 - ET
A: 0s - loss: 8.7173 - acc: 0.448 - ETA: 0s - loss: 8.7268 - acc: 0.448 - ET
A: 0s - loss: 8.7220 - acc: 0.448 - ETA: 0s - loss: 8.7233 - acc: 0.448 - ET
A: 0s - loss: 8.7327 - acc: 0.448 - ETA: 0s - loss: 8.7473 - acc: 0.447 - ET
A: 0s - loss: 8.7479 - acc: 0.447 - ETA: 0s - loss: 8.7599 - acc: 0.446 - ET
A: 0s - loss: 8.7753 - acc: 0.445 - ETA: 0s - loss: 8.7777 - acc: 0.445 - ET
A: 0s - loss: 8.7673 - acc: 0.446 - ETA: 0s - loss: 8.7756 - acc: 0.4453Epoch
00009: val_loss improved from 9.38226 to 9.29182, saving model to saved_model
s/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.7734 - acc: 0.4455
 - val_loss: 9.2918 - val_acc: 0.3749
Epoch 11/20
6540/6680 [============================>.] - ETA: 2s - loss: 11.2829 - acc:
 0.30 - ETA: 2s - loss: 8.7968 - acc: 0.4500 - ETA: 2s - loss: 8.9383 - acc:
 0.443 - ETA: 2s - loss: 8.6599 - acc: 0.461 - ETA: 2s - loss: 8.5605 - acc:
 0.467 - ETA: 2s - loss: 8.4833 - acc: 0.472 - ETA: 2s - loss: 8.4167 - acc:
 0.476 - ETA: 2s - loss: 8.5361 - acc: 0.468 - ETA: 2s - loss: 8.5964 - acc:
 0.464 - ETA: 2s - loss: 8.6119 - acc: 0.462 - ETA: 2s - loss: 8.6611 - acc:
 0.460 - ETA: 2s - loss: 8.6587 - acc: 0.460 - ETA: 1s - loss: 8.7150 - acc:
 0.457 - ETA: 1s - loss: 8.8430 - acc: 0.449 - ETA: 1s - loss: 8.8647 - acc:
 0.447 - ETA: 1s - loss: 8.9438 - acc: 0.442 - ETA: 1s - loss: 8.9245 - acc:
 0.443 - ETA: 1s - loss: 8.9226 - acc: 0.443 - ETA: 1s - loss: 8.8701 - acc:
 0.446 - ETA: 1s - loss: 8.8699 - acc: 0.446 - ETA: 1s - loss: 8.9232 - acc:
 0.443 - ETA: 1s - loss: 8.9441 - acc: 0.442 - ETA: 1s - loss: 8.9031 - acc:
 0.444 - ETA: 1s - loss: 8.8976 - acc: 0.444 - ETA: 1s - loss: 8.9162 - acc:
 0.443 - ETA: 1s - loss: 8.8765 - acc: 0.446 - ETA: 1s - loss: 8.8187 - acc:
 0.449 - ETA: 1s - loss: 8.8079 - acc: 0.450 - ETA: 1s - loss: 8.8269 - acc:
 0.449 - ETA: 1s - loss: 8.7840 - acc: 0.451 - ETA: 1s - loss: 8.7674 - acc:
 0.452 - ETA: 0s - loss: 8.7940 - acc: 0.450 - ETA: 0s - loss: 8.7968 - acc:
 0.449 - ETA: 0s - loss: 8.7922 - acc: 0.450 - ETA: 0s - loss: 8.7806 - acc:
 0.450 - ETA: 0s - loss: 8.7968 - acc: 0.449 - ETA: 0s - loss: 8.8232 - acc:
 0.447 - ETA: 0s - loss: 8.7964 - acc: 0.448 - ETA: 0s - loss: 8.7888 - acc:
 0.449 - ETA: 0s - loss: 8.7787 - acc: 0.450 - ETA: 0s - loss: 8.7763 - acc:
 0.449 - ETA: 0s - loss: 8.7937 - acc: 0.448 - ETA: 0s - loss: 8.7790 - acc:
 0.449 - ETA: 0s - loss: 8.7496 - acc: 0.450 - ETA: 0s - loss: 8.7300 - acc:
 0.451 - ETA: 0s - loss: 8.7438 - acc: 0.450 - ETA: 0s - loss: 8.7213 - acc:
 0.451 - ETA: 0s - loss: 8.7370 - acc: 0.450 - ETA: 0s - loss: 8.7224 - acc:
 0.4511Epoch 00010: val_loss improved from 9.29182 to 9.20740, saving model t
o saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.7087 - acc: 0.4521
 - val_loss: 9.2074 - val_acc: 0.3916
Epoch 12/20
6640/6680 [============================>.] - ETA: 2s - loss: 10.4773 - acc:
 0.35 - ETA: 2s - loss: 9.9381 - acc: 0.3750 - ETA: 2s - loss: 9.0365 - acc:
 0.426 - ETA: 2s - loss: 8.5663 - acc: 0.456 - ETA: 2s - loss: 8.5277 - acc:
 0.462 - ETA: 2s - loss: 8.5263 - acc: 0.463 - ETA: 2s - loss: 8.5289 - acc:
 0.464 - ETA: 2s - loss: 8.5285 - acc: 0.464 - ETA: 2s - loss: 8.4713 - acc:
 0.466 - ETA: 2s - loss: 8.6178 - acc: 0.457 - ETA: 2s - loss: 8.6284 - acc:
 0.458 - ETA: 2s - loss: 8.5947 - acc: 0.460 - ETA: 2s - loss: 8.6623 - acc:
 0.456 - ETA: 1s - loss: 8.6269 - acc: 0.458 - ETA: 1s - loss: 8.5911 - acc:
 0.461 - ETA: 1s - loss: 8.5660 - acc: 0.462 - ETA: 1s - loss: 8.6740 - acc:
 0.456 - ETA: 1s - loss: 8.6858 - acc: 0.455 - ETA: 1s - loss: 8.6963 - acc:
```

```
 0.455 - ETA: 1s - loss: 8.6631 - acc: 0.457 - ETA: 1s - loss: 8.6561 - acc:
 0.457 - ETA: 1s - loss: 8.6607 - acc: 0.457 - ETA: 1s - loss: 8.6597 - acc:
 0.457 - ETA: 1s - loss: 8.7270 - acc: 0.453 - ETA: 1s - loss: 8.6851 - acc:
 0.456 - ETA: 1s - loss: 8.6244 - acc: 0.459 - ETA: 1s - loss: 8.6009 - acc:
 0.460 - ETA: 1s - loss: 8.6066 - acc: 0.460 - ETA: 1s - loss: 8.5628 - acc:
 0.462 - ETA: 1s - loss: 8.5855 - acc: 0.461 - ETA: 0s - loss: 8.5882 - acc:
 0.461 - ETA: 0s - loss: 8.5861 - acc: 0.461 - ETA: 0s - loss: 8.5844 - acc:
 0.461 - ETA: 0s - loss: 8.5550 - acc: 0.463 - ETA: 0s - loss: 8.5334 - acc:
 0.464 - ETA: 0s - loss: 8.5436 - acc: 0.464 - ETA: 0s - loss: 8.5310 - acc:
 0.465 - ETA: 0s - loss: 8.5290 - acc: 0.465 - ETA: 0s - loss: 8.5411 - acc:
 0.464 - ETA: 0s - loss: 8.5630 - acc: 0.462 - ETA: 0s - loss: 8.5650 - acc:
 0.462 - ETA: 0s - loss: 8.5729 - acc: 0.462 - ETA: 0s - loss: 8.5938 - acc:
 0.461 - ETA: 0s - loss: 8.5793 - acc: 0.462 - ETA: 0s - loss: 8.5843 - acc:
 0.461 - ETA: 0s - loss: 8.6141 - acc: 0.459 - ETA: 0s - loss: 8.6240 - acc:
 0.459 - ETA: 0s - loss: 8.6269 - acc: 0.459 - ETA: 0s - loss: 8.6282 - acc:
 0.4590Epoch 00011: val_loss improved from 9.20740 to 9.13604, saving model t
o saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.6297 - acc: 0.4590
 - val_loss: 9.1360 - val_acc: 0.3904
Epoch 13/20
6560/6680 [============================>.] - ETA: 3s - loss: 6.4589 - acc: 0.
600 - ETA: 2s - loss: 8.5719 - acc: 0.462 - ETA: 2s - loss: 7.7776 - acc: 0.5
14 - ETA: 2s - loss: 8.3938 - acc: 0.475 - ETA: 2s - loss: 8.4566 - acc: 0.47
2 - ETA: 2s - loss: 8.5882 - acc: 0.464 - ETA: 2s - loss: 8.5775 - acc: 0.465
- ETA: 2s - loss: 8.6029 - acc: 0.464 - ETA: 2s - loss: 8.5931 - acc: 0.465 -
ETA: 2s - loss: 8.6517 - acc: 0.461 - ETA: 2s - loss: 8.8129 - acc: 0.451 - E
TA: 2s - loss: 8.7332 - acc: 0.456 - ETA: 1s - loss: 8.6473 - acc: 0.462 - ET
A: 1s - loss: 8.6377 - acc: 0.462 - ETA: 1s - loss: 8.5882 - acc: 0.465 - ET
A: 1s - loss: 8.5140 - acc: 0.470 - ETA: 1s - loss: 8.5217 - acc: 0.469 - ET
A: 1s - loss: 8.5227 - acc: 0.469 - ETA: 1s - loss: 8.5295 - acc: 0.468 - ET
A: 1s - loss: 8.5383 - acc: 0.467 - ETA: 1s - loss: 8.5145 - acc: 0.468 - ET
A: 1s - loss: 8.5044 - acc: 0.469 - ETA: 1s - loss: 8.5423 - acc: 0.466 - ET
A: 1s - loss: 8.5971 - acc: 0.463 - ETA: 1s - loss: 8.6373 - acc: 0.460 - ET
A: 1s - loss: 8.6467 - acc: 0.460 - ETA: 1s - loss: 8.6336 - acc: 0.461 - ET
A: 1s - loss: 8.6466 - acc: 0.460 - ETA: 1s - loss: 8.6338 - acc: 0.461 - ET
A: 1s - loss: 8.6569 - acc: 0.460 - ETA: 0s - loss: 8.6602 - acc: 0.459 - ET
A: 0s - loss: 8.6332 - acc: 0.461 - ETA: 0s - loss: 8.6284 - acc: 0.461 - ET
A: 0s - loss: 8.6224 - acc: 0.461 - ETA: 0s - loss: 8.6235 - acc: 0.461 - ET
A: 0s - loss: 8.6403 - acc: 0.460 - ETA: 0s - loss: 8.6522 - acc: 0.459 - ET
A: 0s - loss: 8.6530 - acc: 0.459 - ETA: 0s - loss: 8.5936 - acc: 0.463 - ET
A: 0s - loss: 8.6017 - acc: 0.462 - ETA: 0s - loss: 8.5742 - acc: 0.464 - ET
A: 0s - loss: 8.5855 - acc: 0.463 - ETA: 0s - loss: 8.5860 - acc: 0.463 - ET
A: 0s - loss: 8.5718 - acc: 0.463 - ETA: 0s - loss: 8.5631 - acc: 0.464 - ET
A: 0s - loss: 8.5673 - acc: 0.463 - ETA: 0s - loss: 8.5575 - acc: 0.464 - ET
A: 0s - loss: 8.5550 - acc: 0.4642Epoch 00012: val_loss improved from 9.13604
to 8.98509, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.5582 - acc: 0.4641
 - val_loss: 8.9851 - val_acc: 0.4000
Epoch 14/20
6580/6680 [============================>.] - ETA: 2s - loss: 8.8694 - acc: 0.
450 - ETA: 2s - loss: 8.8690 - acc: 0.450 - ETA: 2s - loss: 8.2231 - acc: 0.4
90 - ETA: 2s - loss: 8.4480 - acc: 0.475 - ETA: 2s - loss: 8.6601 - acc: 0.46
2 - ETA: 2s - loss: 8.6490 - acc: 0.462 - ETA: 2s - loss: 8.5575 - acc: 0.467
- ETA: 2s - loss: 8.4810 - acc: 0.471 - ETA: 2s - loss: 8.4508 - acc: 0.473 -
ETA: 2s - loss: 8.5543 - acc: 0.467 - ETA: 2s - loss: 8.5295 - acc: 0.467 - E
TA: 1s - loss: 8.4671 - acc: 0.470 - ETA: 1s - loss: 8.4843 - acc: 0.469 - ET
A: 1s - loss: 8.4693 - acc: 0.470 - ETA: 1s - loss: 8.5072 - acc: 0.467 - ET
```

```
A: 1s - loss: 8.5270 - acc: 0.466 - ETA: 1s - loss: 8.5551 - acc: 0.464 - ET
A: 1s - loss: 8.5440 - acc: 0.465 - ETA: 1s - loss: 8.4744 - acc: 0.469 - ET
A: 1s - loss: 8.4677 - acc: 0.470 - ETA: 1s - loss: 8.4557 - acc: 0.471 - ET
A: 1s - loss: 8.5007 - acc: 0.468 - ETA: 1s - loss: 8.5033 - acc: 0.468 - ET
A: 1s - loss: 8.5312 - acc: 0.466 - ETA: 1s - loss: 8.5595 - acc: 0.464 - ET
A: 1s - loss: 8.5062 - acc: 0.467 - ETA: 1s - loss: 8.5099 - acc: 0.466 - ET
A: 1s - loss: 8.4925 - acc: 0.468 - ETA: 1s - loss: 8.4820 - acc: 0.468 - ET
A: 1s - loss: 8.4875 - acc: 0.468 - ETA: 1s - loss: 8.4961 - acc: 0.468 - ET
A: 0s - loss: 8.4586 - acc: 0.470 - ETA: 0s - loss: 8.4591 - acc: 0.470 - ET
A: 0s - loss: 8.4791 - acc: 0.469 - ETA: 0s - loss: 8.4909 - acc: 0.467 - ET
A: 0s - loss: 8.4848 - acc: 0.467 - ETA: 0s - loss: 8.4807 - acc: 0.467 - ET
A: 0s - loss: 8.5071 - acc: 0.465 - ETA: 0s - loss: 8.5162 - acc: 0.465 - ET
A: 0s - loss: 8.4890 - acc: 0.466 - ETA: 0s - loss: 8.5024 - acc: 0.465 - ET
A: 0s - loss: 8.5044 - acc: 0.465 - ETA: 0s - loss: 8.5031 - acc: 0.465 - ET
A: 0s - loss: 8.4994 - acc: 0.465 - ETA: 0s - loss: 8.4713 - acc: 0.466 - ET
A: 0s - loss: 8.4498 - acc: 0.468 - ETA: 0s - loss: 8.4579 - acc: 0.467 - ET
A: 0s - loss: 8.4472 - acc: 0.468 - ETA: 0s - loss: 8.4351 - acc: 0.469 - ET
A: 0s - loss: 8.4441 - acc: 0.4687Epoch 00013: val_loss improved from 8.98509
to 8.88833, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.4568 - acc: 0.4680
 - val_loss: 8.8883 - val_acc: 0.4024
Epoch 15/20
6660/6680 [============================>.] - ETA: 2s - loss: 10.5237 - acc:
 0.30 - ETA: 2s - loss: 8.4946 - acc: 0.4625 - ETA: 2s - loss: 8.4070 - acc:
 0.470 - ETA: 2s - loss: 8.5571 - acc: 0.461 - ETA: 2s - loss: 8.2339 - acc:
 0.479 - ETA: 2s - loss: 8.4144 - acc: 0.468 - ETA: 2s - loss: 8.5745 - acc:
 0.458 - ETA: 2s - loss: 8.4790 - acc: 0.464 - ETA: 2s - loss: 8.5687 - acc:
 0.459 - ETA: 2s - loss: 8.5183 - acc: 0.462 - ETA: 2s - loss: 8.5607 - acc:
 0.458 - ETA: 2s - loss: 8.5610 - acc: 0.458 - ETA: 1s - loss: 8.5361 - acc:
 0.459 - ETA: 1s - loss: 8.5219 - acc: 0.461 - ETA: 1s - loss: 8.4858 - acc:
 0.464 - ETA: 1s - loss: 8.4976 - acc: 0.463 - ETA: 1s - loss: 8.5124 - acc:
 0.462 - ETA: 1s - loss: 8.4777 - acc: 0.464 - ETA: 1s - loss: 8.5374 - acc:
 0.460 - ETA: 1s - loss: 8.4884 - acc: 0.462 - ETA: 1s - loss: 8.5156 - acc:
 0.460 - ETA: 1s - loss: 8.5139 - acc: 0.460 - ETA: 1s - loss: 8.5144 - acc:
 0.461 - ETA: 1s - loss: 8.5094 - acc: 0.461 - ETA: 1s - loss: 8.5103 - acc:
 0.461 - ETA: 1s - loss: 8.5242 - acc: 0.461 - ETA: 1s - loss: 8.5200 - acc:
 0.461 - ETA: 1s - loss: 8.4842 - acc: 0.463 - ETA: 1s - loss: 8.4746 - acc:
 0.464 - ETA: 1s - loss: 8.4721 - acc: 0.464 - ETA: 1s - loss: 8.4676 - acc:
 0.465 - ETA: 0s - loss: 8.4424 - acc: 0.467 - ETA: 0s - loss: 8.4379 - acc:
 0.467 - ETA: 0s - loss: 8.4333 - acc: 0.467 - ETA: 0s - loss: 8.3981 - acc:
 0.469 - ETA: 0s - loss: 8.3757 - acc: 0.471 - ETA: 0s - loss: 8.3610 - acc:
 0.471 - ETA: 0s - loss: 8.3472 - acc: 0.472 - ETA: 0s - loss: 8.3468 - acc:
 0.472 - ETA: 0s - loss: 8.3725 - acc: 0.471 - ETA: 0s - loss: 8.3745 - acc:
 0.471 - ETA: 0s - loss: 8.3502 - acc: 0.472 - ETA: 0s - loss: 8.3331 - acc:
 0.473 - ETA: 0s - loss: 8.3156 - acc: 0.475 - ETA: 0s - loss: 8.3123 - acc:
 0.475 - ETA: 0s - loss: 8.3329 - acc: 0.474 - ETA: 0s - loss: 8.3116 - acc:
 0.475 - ETA: 0s - loss: 8.3291 - acc: 0.474 - ETA: 0s - loss: 8.3285 - acc:
 0.474 - ETA: 0s - loss: 8.3527 - acc: 0.4734Epoch 00014: val_loss did not im
prove
6680/6680 [==============================] - 2s - loss: 8.3663 - acc: 0.4726
 - val_loss: 8.9176 - val_acc: 0.3988
Epoch 16/20
6560/6680 [============================>.] - ETA: 2s - loss: 8.0591 - acc: 0.
500 - ETA: 2s - loss: 8.6683 - acc: 0.456 - ETA: 2s - loss: 8.0711 - acc: 0.4
92 - ETA: 2s - loss: 8.5670 - acc: 0.462 - ETA: 2s - loss: 8.6792 - acc: 0.45
3 - ETA: 2s - loss: 8.6982 - acc: 0.453 - ETA: 2s - loss: 8.7538 - acc: 0.447
- ETA: 2s - loss: 8.7335 - acc: 0.450 - ETA: 2s - loss: 8.6024 - acc: 0.457 -
```

```
ETA: 2s - loss: 8.7624 - acc: 0.447 - ETA: 2s - loss: 8.7788 - acc: 0.446 - E
TA: 2s - loss: 8.7452 - acc: 0.447 - ETA: 2s - loss: 8.6891 - acc: 0.450 - ET
A: 2s - loss: 8.6778 - acc: 0.451 - ETA: 1s - loss: 8.6258 - acc: 0.454 - ET
A: 1s - loss: 8.6032 - acc: 0.456 - ETA: 1s - loss: 8.5601 - acc: 0.459 - ET
A: 1s - loss: 8.5050 - acc: 0.463 - ETA: 1s - loss: 8.5108 - acc: 0.463 - ET
A: 1s - loss: 8.4999 - acc: 0.464 - ETA: 1s - loss: 8.5492 - acc: 0.461 - ET
A: 1s - loss: 8.5168 - acc: 0.463 - ETA: 1s - loss: 8.4954 - acc: 0.465 - ET
A: 1s - loss: 8.5017 - acc: 0.465 - ETA: 1s - loss: 8.4397 - acc: 0.469 - ET
A: 1s - loss: 8.4684 - acc: 0.467 - ETA: 1s - loss: 8.4318 - acc: 0.470 - ET
A: 1s - loss: 8.3718 - acc: 0.473 - ETA: 1s - loss: 8.3918 - acc: 0.472 - ET
A: 1s - loss: 8.3842 - acc: 0.473 - ETA: 1s - loss: 8.3863 - acc: 0.472 - ET
A: 0s - loss: 8.3435 - acc: 0.475 - ETA: 0s - loss: 8.3720 - acc: 0.473 - ET
A: 0s - loss: 8.3840 - acc: 0.473 - ETA: 0s - loss: 8.3953 - acc: 0.472 - ET
A: 0s - loss: 8.3691 - acc: 0.473 - ETA: 0s - loss: 8.3362 - acc: 0.475 - ET
A: 0s - loss: 8.3381 - acc: 0.475 - ETA: 0s - loss: 8.3213 - acc: 0.477 - ET
A: 0s - loss: 8.3092 - acc: 0.477 - ETA: 0s - loss: 8.3266 - acc: 0.476 - ET
A: 0s - loss: 8.3261 - acc: 0.476 - ETA: 0s - loss: 8.3229 - acc: 0.477 - ET
A: 0s - loss: 8.3221 - acc: 0.476 - ETA: 0s - loss: 8.3088 - acc: 0.477 - ET
A: 0s - loss: 8.3193 - acc: 0.476 - ETA: 0s - loss: 8.3292 - acc: 0.476 - ET
A: 0s - loss: 8.3081 - acc: 0.477 - ETA: 0s - loss: 8.3250 - acc: 0.4768Epoch
00015: val_loss improved from 8.88833 to 8.84847, saving model to saved_model
s/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.3154 - acc: 0.4775
 - val_loss: 8.8485 - val_acc: 0.4012
Epoch 17/20
6600/6680 [============================>.] - ETA: 2s - loss: 8.4524 - acc: 0.
450 - ETA: 2s - loss: 7.8063 - acc: 0.512 - ETA: 2s - loss: 8.2841 - acc: 0.4
84 - ETA: 2s - loss: 8.6986 - acc: 0.458 - ETA: 2s - loss: 9.0375 - acc: 0.43
6 - ETA: 2s - loss: 8.9652 - acc: 0.440 - ETA: 2s - loss: 8.7330 - acc: 0.454
- ETA: 2s - loss: 8.6348 - acc: 0.459 - ETA: 2s - loss: 8.6488 - acc: 0.459 -
ETA: 2s - loss: 8.5753 - acc: 0.463 - ETA: 1s - loss: 8.5141 - acc: 0.468 - E
TA: 1s - loss: 8.6271 - acc: 0.461 - ETA: 1s - loss: 8.7027 - acc: 0.457 - ET
A: 1s - loss: 8.7221 - acc: 0.456 - ETA: 1s - loss: 8.6514 - acc: 0.460 - ET
A: 1s - loss: 8.5929 - acc: 0.463 - ETA: 1s - loss: 8.6010 - acc: 0.461 - ET
A: 1s - loss: 8.5580 - acc: 0.464 - ETA: 1s - loss: 8.5369 - acc: 0.466 - ET
A: 1s - loss: 8.5821 - acc: 0.463 - ETA: 1s - loss: 8.5622 - acc: 0.464 - ET
A: 1s - loss: 8.5377 - acc: 0.466 - ETA: 1s - loss: 8.4955 - acc: 0.469 - ET
A: 1s - loss: 8.4323 - acc: 0.472 - ETA: 1s - loss: 8.3883 - acc: 0.475 - ET
A: 1s - loss: 8.3960 - acc: 0.475 - ETA: 1s - loss: 8.3877 - acc: 0.475 - ET
A: 1s - loss: 8.3969 - acc: 0.475 - ETA: 1s - loss: 8.3808 - acc: 0.476 - ET
A: 1s - loss: 8.3301 - acc: 0.479 - ETA: 0s - loss: 8.3412 - acc: 0.478 - ET
A: 0s - loss: 8.3471 - acc: 0.478 - ETA: 0s - loss: 8.3502 - acc: 0.478 - ET
A: 0s - loss: 8.3343 - acc: 0.479 - ETA: 0s - loss: 8.3339 - acc: 0.479 - ET
A: 0s - loss: 8.3492 - acc: 0.478 - ETA: 0s - loss: 8.3636 - acc: 0.477 - ET
A: 0s - loss: 8.3553 - acc: 0.478 - ETA: 0s - loss: 8.3597 - acc: 0.478 - ET
A: 0s - loss: 8.3530 - acc: 0.478 - ETA: 0s - loss: 8.3254 - acc: 0.480 - ET
A: 0s - loss: 8.2991 - acc: 0.482 - ETA: 0s - loss: 8.2934 - acc: 0.482 - ET
A: 0s - loss: 8.3124 - acc: 0.481 - ETA: 0s - loss: 8.2950 - acc: 0.482 - ET
A: 0s - loss: 8.2932 - acc: 0.482 - ETA: 0s - loss: 8.2989 - acc: 0.482 - ET
A: 0s - loss: 8.2995 - acc: 0.481 - ETA: 0s - loss: 8.2897 - acc: 0.4824Epoch
00016: val_loss improved from 8.84847 to 8.80827, saving model to saved_model
s/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.2993 - acc: 0.4817
 - val_loss: 8.8083 - val_acc: 0.4120
Epoch 18/20
6660/6680 [============================>.] - ETA: 2s - loss: 8.8650 - acc: 0.
450 - ETA: 2s - loss: 9.3188 - acc: 0.418 - ETA: 2s - loss: 8.5659 - acc: 0.4
```

```
63 - ETA: 2s - loss: 8.6600 - acc: 0.459 - ETA: 2s - loss: 8.4168 - acc: 0.47
5 - ETA: 2s - loss: 8.3229 - acc: 0.481 - ETA: 2s - loss: 8.3066 - acc: 0.481
- ETA: 2s - loss: 8.4888 - acc: 0.470 - ETA: 2s - loss: 8.4635 - acc: 0.472 -
ETA: 2s - loss: 8.4660 - acc: 0.471 - ETA: 2s - loss: 8.4590 - acc: 0.472 - E
TA: 2s - loss: 8.5253 - acc: 0.467 - ETA: 1s - loss: 8.5127 - acc: 0.467 - ET
A: 1s - loss: 8.5465 - acc: 0.464 - ETA: 1s - loss: 8.5955 - acc: 0.461 - ET
A: 1s - loss: 8.5747 - acc: 0.461 - ETA: 1s - loss: 8.5349 - acc: 0.464 - ET
A: 1s - loss: 8.5316 - acc: 0.464 - ETA: 1s - loss: 8.4623 - acc: 0.467 - ET
A: 1s - loss: 8.4139 - acc: 0.470 - ETA: 1s - loss: 8.4271 - acc: 0.469 - ET
A: 1s - loss: 8.4292 - acc: 0.469 - ETA: 1s - loss: 8.4296 - acc: 0.469 - ET
A: 1s - loss: 8.4058 - acc: 0.470 - ETA: 1s - loss: 8.3516 - acc: 0.474 - ET
A: 1s - loss: 8.3681 - acc: 0.472 - ETA: 1s - loss: 8.3794 - acc: 0.472 - ET
A: 1s - loss: 8.4198 - acc: 0.470 - ETA: 1s - loss: 8.4404 - acc: 0.469 - ET
A: 1s - loss: 8.4410 - acc: 0.468 - ETA: 1s - loss: 8.4160 - acc: 0.470 - ET
A: 0s - loss: 8.4153 - acc: 0.471 - ETA: 0s - loss: 8.4121 - acc: 0.471 - ET
A: 0s - loss: 8.4174 - acc: 0.470 - ETA: 0s - loss: 8.3569 - acc: 0.474 - ET
A: 0s - loss: 8.3282 - acc: 0.476 - ETA: 0s - loss: 8.3210 - acc: 0.476 - ET
A: 0s - loss: 8.3256 - acc: 0.476 - ETA: 0s - loss: 8.2969 - acc: 0.478 - ET
A: 0s - loss: 8.2995 - acc: 0.478 - ETA: 0s - loss: 8.3062 - acc: 0.477 - ET
A: 0s - loss: 8.2816 - acc: 0.479 - ETA: 0s - loss: 8.3017 - acc: 0.478 - ET
A: 0s - loss: 8.2826 - acc: 0.479 - ETA: 0s - loss: 8.2552 - acc: 0.480 - ET
A: 0s - loss: 8.2639 - acc: 0.480 - ETA: 0s - loss: 8.2660 - acc: 0.480 - ET
A: 0s - loss: 8.2428 - acc: 0.481 - ETA: 0s - loss: 8.2385 - acc: 0.481 - ET
A: 0s - loss: 8.2203 - acc: 0.4829Epoch 00017: val_loss did not improve
6680/6680 [==============================] - 2s - loss: 8.2228 - acc: 0.4825
 - val_loss: 8.8407 - val_acc: 0.4024
Epoch 19/20
6540/6680 [============================>.] - ETA: 2s - loss: 8.0591 - acc: 0.
500 - ETA: 2s - loss: 7.8364 - acc: 0.506 - ETA: 2s - loss: 7.9124 - acc: 0.4
96 - ETA: 2s - loss: 8.0369 - acc: 0.490 - ETA: 2s - loss: 8.2082 - acc: 0.48
1 - ETA: 2s - loss: 8.1079 - acc: 0.486 - ETA: 2s - loss: 8.2411 - acc: 0.479
- ETA: 2s - loss: 8.2163 - acc: 0.482 - ETA: 2s - loss: 8.2040 - acc: 0.482 -
ETA: 2s - loss: 8.2686 - acc: 0.478 - ETA: 2s - loss: 8.1042 - acc: 0.485 - E
TA: 1s - loss: 8.1149 - acc: 0.484 - ETA: 1s - loss: 8.1090 - acc: 0.485 - ET
A: 1s - loss: 8.1474 - acc: 0.482 - ETA: 1s - loss: 8.1516 - acc: 0.482 - ET
A: 1s - loss: 8.1806 - acc: 0.481 - ETA: 1s - loss: 8.1961 - acc: 0.480 - ET
A: 1s - loss: 8.1892 - acc: 0.481 - ETA: 1s - loss: 8.2540 - acc: 0.477 - ET
A: 1s - loss: 8.2460 - acc: 0.477 - ETA: 1s - loss: 8.1957 - acc: 0.480 - ET
A: 1s - loss: 8.2067 - acc: 0.479 - ETA: 1s - loss: 8.1751 - acc: 0.480 - ET
A: 1s - loss: 8.1913 - acc: 0.479 - ETA: 1s - loss: 8.1288 - acc: 0.484 - ET
A: 1s - loss: 8.1393 - acc: 0.483 - ETA: 1s - loss: 8.1336 - acc: 0.484 - ET
A: 1s - loss: 8.1229 - acc: 0.484 - ETA: 1s - loss: 8.1165 - acc: 0.485 - ET
A: 1s - loss: 8.1121 - acc: 0.485 - ETA: 0s - loss: 8.1218 - acc: 0.485 - ET
A: 0s - loss: 8.1276 - acc: 0.485 - ETA: 0s - loss: 8.0957 - acc: 0.487 - ET
A: 0s - loss: 8.0529 - acc: 0.489 - ETA: 0s - loss: 8.0609 - acc: 0.489 - ET
A: 0s - loss: 8.0546 - acc: 0.489 - ETA: 0s - loss: 8.0329 - acc: 0.491 - ET
A: 0s - loss: 8.0345 - acc: 0.490 - ETA: 0s - loss: 8.0648 - acc: 0.488 - ET
A: 0s - loss: 8.0441 - acc: 0.490 - ETA: 0s - loss: 8.0827 - acc: 0.487 - ET
A: 0s - loss: 8.0896 - acc: 0.487 - ETA: 0s - loss: 8.0810 - acc: 0.487 - ET
A: 0s - loss: 8.0913 - acc: 0.487 - ETA: 0s - loss: 8.0937 - acc: 0.487 - ET
A: 0s - loss: 8.0955 - acc: 0.486 - ETA: 0s - loss: 8.0922 - acc: 0.487 - ET
A: 0s - loss: 8.0915 - acc: 0.4876Epoch 00018: val_loss improved from 8.80827
to 8.69511, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 8.0914 - acc: 0.4877
 - val_loss: 8.6951 - val_acc: 0.4156
Epoch 20/20
6640/6680 [============================>.] - ETA: 2s - loss: 9.6709 - acc: 0.
```

```
400 - ETA: 2s - loss: 8.1905 - acc: 0.485 - ETA: 2s - loss: 7.7839 - acc: 0.5
10 - ETA: 2s - loss: 7.5220 - acc: 0.526 - ETA: 2s - loss: 7.6400 - acc: 0.51
7 - ETA: 2s - loss: 7.4708 - acc: 0.530 - ETA: 2s - loss: 7.8414 - acc: 0.507
- ETA: 2s - loss: 7.9782 - acc: 0.496 - ETA: 2s - loss: 7.9724 - acc: 0.495 -
ETA: 2s - loss: 7.9563 - acc: 0.497 - ETA: 2s - loss: 7.9917 - acc: 0.496 - E
TA: 1s - loss: 8.0460 - acc: 0.493 - ETA: 1s - loss: 8.0665 - acc: 0.492 - ET
A: 1s - loss: 7.9951 - acc: 0.497 - ETA: 1s - loss: 7.9923 - acc: 0.498 - ET
A: 1s - loss: 8.0356 - acc: 0.495 - ETA: 1s - loss: 8.0684 - acc: 0.493 - ET
A: 1s - loss: 8.0737 - acc: 0.491 - ETA: 1s - loss: 8.0992 - acc: 0.490 - ET
A: 1s - loss: 8.0713 - acc: 0.492 - ETA: 1s - loss: 8.0017 - acc: 0.496 - ET
A: 1s - loss: 7.9941 - acc: 0.496 - ETA: 1s - loss: 7.9786 - acc: 0.497 - ET
A: 1s - loss: 7.9876 - acc: 0.497 - ETA: 1s - loss: 8.0292 - acc: 0.494 - ET
A: 1s - loss: 8.0495 - acc: 0.493 - ETA: 1s - loss: 8.0531 - acc: 0.492 - ET
A: 1s - loss: 8.0701 - acc: 0.490 - ETA: 1s - loss: 8.0989 - acc: 0.489 - ET
A: 1s - loss: 8.1225 - acc: 0.488 - ETA: 0s - loss: 8.0952 - acc: 0.489 - ET
A: 0s - loss: 8.1145 - acc: 0.488 - ETA: 0s - loss: 8.0697 - acc: 0.491 - ET
A: 0s - loss: 8.0627 - acc: 0.492 - ETA: 0s - loss: 8.0335 - acc: 0.493 - ET
A: 0s - loss: 8.0380 - acc: 0.493 - ETA: 0s - loss: 8.0045 - acc: 0.495 - ET
A: 0s - loss: 8.0091 - acc: 0.495 - ETA: 0s - loss: 7.9861 - acc: 0.496 - ET
A: 0s - loss: 7.9942 - acc: 0.496 - ETA: 0s - loss: 8.0062 - acc: 0.495 - ET
A: 0s - loss: 8.0022 - acc: 0.496 - ETA: 0s - loss: 7.9842 - acc: 0.497 - ET
A: 0s - loss: 7.9887 - acc: 0.497 - ETA: 0s - loss: 7.9887 - acc: 0.497 - ET
A: 0s - loss: 8.0008 - acc: 0.496 - ETA: 0s - loss: 7.9896 - acc: 0.497 - ET
A: 0s - loss: 7.9791 - acc: 0.498 - ETA: 0s - loss: 7.9617 - acc: 0.4994Epoch
00019: val_loss improved from 8.69511 to 8.57595, saving model to saved_model
s/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s - loss: 7.9647 - acc: 0.4993
 - val_loss: 8.5760 - val_acc: 0.4180
```

```
Out[77]: <keras.callbacks.History at 0x13e44126f98>
```

Load the Model with the Best Validation Loss

```
In [78]: VGG16_model.load_weights('saved_models/weights.best.VGG16.hdf5')
```

Test the Model

Now, we can use the CNN to test how well it identifies breed within our test dataset of dog images. We print the test accuracy below.

```
In [79]: # get index of predicted dog breed for each image in test set
VGG16_predictions = [np.argmax(VGG16_model.predict(np.expand_dims(feature, axi
s=0))) for feature in test_VGG16]

# report test accuracy
test_accuracy =
100*np.sum(np.array(VGG16_predictions)==np.argmax(test_targets, axis=1))/len(V
GG16_predictions)
print('VGG16 model, Test accuracy: %.4f%%' % test_accuracy)
```

```
VGG16 model, Test accuracy: 40.5502%
```

Predict Dog Breed with the Model

In [80]:
```
from extract_bottleneck_features import *

def VGG16_predict_breed(img_path):
    # extract bottleneck features
    bottleneck_feature = extract_VGG16(path_to_tensor(img_path))
    # obtain predicted vector
    predicted_vector = VGG16_model.predict(bottleneck_feature)
    # return dog breed that is predicted by the model
    return dog_names[np.argmax(predicted_vector)]
```

Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning)

You will now use transfer learning to create a CNN that can identify dog breed from images. Your CNN must attain at least 60% accuracy on the test set.

In Step 4, we used transfer learning to create a CNN using VGG-16 bottleneck features. In this section, you must use the bottleneck features from a different pre-trained model. To make things easier for you, we have pre-computed the features for all of the networks that are currently available in Keras:

- VGG-19 (https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/DogVGG19Data.npz) bottleneck features
- ResNet-50 (https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/DogResnet50Data.npz) bottleneck features
- Inception (https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/DogInceptionV3Data.npz) bottleneck features
- Xception (https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/DogXceptionData.npz) bottleneck features

The files are encoded as such:

```
Dog{network}Data.npz
```

where `{network}`, in the above filename, can be one of `VGG19`, `Resnet50`, `InceptionV3`, or `Xception`. Pick one of the above architectures, download the corresponding bottleneck features, and store the downloaded file in the `bottleneck_features/` folder in the repository.

(IMPLEMENTATION) Obtain Bottleneck Features

In the code block below, extract the bottleneck features corresponding to the train, test, and validation sets by running the following:

```
bottleneck_features = np.load('bottleneck_features/Dog{network}Data.npz')
train_{network} = bottleneck_features['train']
valid_{network} = bottleneck_features['valid']
test_{network} = bottleneck_features['test']
```

In [46]:
```
### TODO: Obtain bottleneck features from another pre-trained CNN.
my_bottleneck_features = np.load('bottleneck_features/DogResnet50Data.npz')
my_train_ResNet50 = my_bottleneck_features['train']
my_valid_ResNet50 = my_bottleneck_features['valid']
my_test_ResNet50 = my_bottleneck_features['test']
```

(IMPLEMENTATION) Model Architecture

Create a CNN to classify dog breed. At the end of your code cell block, summarize the layers of your model by executing the line:

```
<your model's name>.summary()
```

Question 5: Outline the steps you took to get to your final CNN architecture and your reasoning at each step. Describe why you think the architecture is suitable for the current problem.

Answer:

Using a pretrained CNN is different than a CNN from scratch. Initially attempts were made to add additional Convolutional layers on top of the Resnet model, but the results were not as good as just adding Densly connected layers to the end of the resnet model. This is most likely because the pre trained resnet50 model was trained on images that are similar to the dog classification images I used for my training. The maximum accuraccy obtained using the pretrained resnet50 as a base, was 86%. I could not achieve a higher accuracy, likely because of limitations of the pretrained resnet50 model.

Various sizes and depth networks were tried, including no networks, single small dense networks, small deep networks, and large networks.

The optimal solution involved adding a dropout to the beginning of the network, which randomly ignores some of the original features at each ephoch. This increases the training focus range.

Additionally, Using 2 very large Dense networks, both with high dropout, avoids memorization, and equally distributes the gradients across a broader range of nodes in the fully connected layers. This creates a normalized attention to various segments of the images linked to their respective activated features.

Training was modified to allow 'resetting' the network to a network that has not diverged from decent test accuracy too much. This means that training was done in a loop, with early_stopping, and then the network was reset to the most recent set of good weights, and then training resumed.

Learning rate reduction was utilized, and this works better when an initially lower learning rate is compiled into the optimizer (.0001). This increases training time only slightly, but offers a smoother training, avoiding training overshooting.

In [29]:
```
### TODO: Define your architecture.


my_ResNet50_model = Sequential()
my_ResNet50_model.add(Dropout(.25,input_shape=my_train_ResNet50.shape[1:]))
my_ResNet50_model.add(GlobalAveragePooling2D())
my_ResNet50_model.add(Dense(2000, activation='relu'))
my_ResNet50_model.add(Dropout(.6))
my_ResNet50_model.add(Dense(2000, activation='relu'))
my_ResNet50_model.add(Dropout(.6))
my_ResNet50_model.add(Dense(133, activation='softmax'))

my_ResNet50_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dropout_2 (Dropout)          (None, 1, 1, 2048)        0         
_________________________________________________________________
global_average_pooling2d_3 ( (None, 2048)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 2000)              4098000   
_________________________________________________________________
dropout_3 (Dropout)          (None, 2000)              0         
_________________________________________________________________
dense_7 (Dense)              (None, 2000)              4002000   
_________________________________________________________________
dropout_4 (Dropout)          (None, 2000)              0         
_________________________________________________________________
dense_8 (Dense)              (None, 133)               266133    
=================================================================
Total params: 8,366,133.0
Trainable params: 8,366,133.0
Non-trainable params: 0.0
_________________________________________________________________
```

Additional Model attempts

```
In [47]:  # 84.92
           # my_ResNet50_model = Sequential()
           # my_ResNet50_model.add(GlobalAveragePooling2D(input_shape=my_train_ResNet50.s
           hape[1:]))
           # my_ResNet50_model.add(Dense(4000,kernel_initializer='he_normal', activation
           ='relu'))
           # my_ResNet50_model.add(Dropout(.65))
           # my_ResNet50_model.add(Dense(133,kernel_initializer='he_normal', activation
           ='softmax'))

           # my_ResNet50_model.summary()


           # 84.8
           # my_ResNet50_model = Sequential()
           # my_ResNet50_model.add(GlobalAveragePooling2D(input_shape=my_train_ResNet50.s
           hape[1:]))
           # my_ResNet50_model.add(Dense(3000, activation='relu'))
           # my_ResNet50_model.add(Dropout(.8))
           # my_ResNet50_model.add(Dense(133, activation='softmax'))

           # my_ResNet50_model.summary()


           # 84.4
           # my_ResNet50_model = Sequential()
           # my_ResNet50_model.add(GlobalAveragePooling2D(input_shape=my_train_ResNet50.s
           hape[1:]))
           # my_ResNet50_model.add(Dense(3000, activation='relu'))
           # my_ResNet50_model.add(Dropout(.8))
           # my_ResNet50_model.add(Dense(133, activation='softmax'))

           # my_ResNet50_model.summary()

           # 84.3
           # my_ResNet50_model = Sequential()
           # my_ResNet50_model.add(GlobalAveragePooling2D(input_shape=my_train_ResNet50.s
           hape[1:]))
           # my_ResNet50_model.add(Dense(500,activation='relu'))
           # my_ResNet50_model.add(Dropout(.7))
           # my_ResNet50_model.add(Dense(133, activation='softmax'))
           # my_ResNet50_model.summary()

           # 83
           # my_ResNet50_model.add(GlobalAveragePooling2D(input_shape=my_train_ResNet50.s
           hape[1:]))
           # my_ResNet50_model.add(Dense(1000, activation='relu'))
           # my_ResNet50_model.add(Dropout(.5))
           # my_ResNet50_model.add(Dense(500, activation='relu'))
           # my_ResNet50_model.add(Dropout(.3))
           # my_ResNet50_model.add(Dense(266, activation='relu'))
           # my_ResNet50_model.add(Dropout(.2))
           # my_ResNet50_model.add(Dense(133, activation='softmax'))

           # my_ResNet50_model.summary()

           #81.3%
           # my_ResNet50_model = Sequential()
           # my_ResNet50_model.add(GlobalAveragePooling2D(input_shape=my_train_ResNet50.s
           hape[1:]))
           # my_ResNet50_model.add(Dense(133, activation='softmax'))
           # my_ResNet50_model.summary()
```

(IMPLEMENTATION) Compile the Model

In [48]:
```
### TODO: Compile the model.
# my_ResNet50_model.compile(loss='categorical_crossentropy', optimizer=RMSprop
(lr=.00001), metrics=['accuracy'])
my_ResNet50_model.compile(loss='categorical_crossentropy', optimizer=RMSprop(l
r=.0001), metrics=['accuracy'])
```

(IMPLEMENTATION) Train the Model

Train your model in the code cell below. Use model checkpointing to save the model that attains the best validation loss.

You are welcome to augment the training data (https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html), but this is not a requirement.

In [49]:
```
### TODO: Train the model.

load_trained_model=True

if load_trained_model:
    load_model("resnet50mod.h5")
else:
    early_stopping = EarlyStopping(monitor='val_loss', patience=20)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.01,
                                  patience=15,  verbose=1)
    checkpointer = ModelCheckpoint(monitor='val_loss',filepath='saved_models/w
eights.best.ResNet50v3.hdf5',
                                   verbose=1, save_best_only=True)

    my_ResNet50_model.fit(my_train_ResNet50, train_targets,
              validation_data=(my_valid_ResNet50, valid_targets),
              epochs=800, batch_size=300, callbacks=[checkpointer,
                                                     TQDMNotebookCallback(leav
e_inner=True,leave_outer=True),

early_stopping,reduce_lr], verbose=0)
    for i in range(4):
        my_ResNet50_model.load_weights('saved_models/weights.best.ResNet50v3.h
df5')
        early_stopping = EarlyStopping(monitor='val_loss', patience=10)
        reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.01,
                                      patience=5,  verbose=1)
        checkpointer = ModelCheckpoint(monitor='val_loss',filepath='saved_mode
ls/weights.best.ResNet50v3.hdf5',
                                       verbose=1, save_best_only=True)

        my_ResNet50_model.fit(my_train_ResNet50, train_targets,
                  validation_data=(my_valid_ResNet50, valid_targets),
                  epochs=800, batch_size=300, callbacks=[checkpointer,

TQDMNotebookCallback(leave_inner=True,leave_outer=True),
                                                         early_stopping,reduce
_lr], verbose=0)
```

(IMPLEMENTATION) Load the Model with the Best Validation Loss

In [33]:
```
### TODO: Load the model weights with the best validation loss.
my_ResNet50_model.load_weights('saved_models/weights.best.ResNet50v3.hdf5')
```

(IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Ensure that your test accuracy is greater than 60%.

In [50]:
```
### TODO: Calculate classification accuracy on the test dataset.
my_ResNet50_predictions =
[np.argmax(my_ResNet50_model.predict(np.expand_dims(feature, axis=0))) for fea
ture in my_test_ResNet50]

# report test accuracy
test_accuracy = 100*np.sum(np.array(my_ResNet50_predictions)==np.argmax(test_t
argets, axis=1))/len(my_ResNet50_predictions)
print('Test accuracy: %.4f%%' % test_accuracy)
```

```
Test accuracy: 85.0478%
```

In [35]:
```
my_ResNet50_model.save("resnet50mod.h5")
```

(IMPLEMENTATION) Predict Dog Breed with the Model

Write a function that takes an image path as input and returns the dog breed (`Affenpinscher`, `Afghan_hound`, etc) that is predicted by your model.

Similar to the analogous function in Step 5, your function should have three steps:

1. Extract the bottleneck features corresponding to the chosen CNN model.
2. Supply the bottleneck features as input to the model to return the predicted vector. Note that the argmax of this prediction vector gives the index of the predicted dog breed.
3. Use the `dog_names` array defined in Step 0 of this notebook to return the corresponding breed.

The functions to extract the bottleneck features can be found in `extract_bottleneck_features.py`, and they have been imported in an earlier code cell. To obtain the bottleneck features corresponding to your chosen CNN architecture, you need to use the function

```
extract_{network}
```

where `{network}`, in the above filename, should be one of `VGG19`, `Resnet50`, `InceptionV3`, or `Xception`.

In [38]:
```
### TODO: Write a function that takes a path to an image as input
### and returns the dog breed that is predicted by the model.
import os
from extract_bottleneck_features import *


def my_ResNet50_predict_breed(img_path):
    # extract bottleneck features
    bottleneck_feature = extract_Resnet50(path_to_tensor(img_path))
    # obtain predicted vector
    predicted_vector = my_ResNet50_model.predict(bottleneck_feature)
    # return dog breed that is predicted by the model
    return dog_names[np.argmax(predicted_vector)]

Brittany_test_files = filter(lambda x: x.endswith('.jpg'), os.listdir('dogImag
es/test/037.Brittany/'))

for file in Brittany_test_files:

predicted_breed=my_ResNet50_predict_breed(('dogImages/test/037.Brittany/'+file

    print("processing file:",file,', Expected Brittany, predicted:',predicted_
breed)
```

```
processing file: Brittany_02591.jpg , Expected Brittany, predicted: Brittany
processing file: Brittany_02601.jpg , Expected Brittany, predicted: Brittany
processing file: Brittany_02607.jpg , Expected Brittany, predicted: Brittany
processing file: Brittany_02622.jpg , Expected Brittany, predicted: Brittany
processing file: Brittany_02633.jpg , Expected Brittany, predicted: Brittany
processing file: Brittany_02648.jpg , Expected Brittany, predicted: Brittany
```

In [52]:
```
my_ResNet50_predict_breed(('our_brittany.jpg'))
```

Out[52]:
```
'Brittany'
```

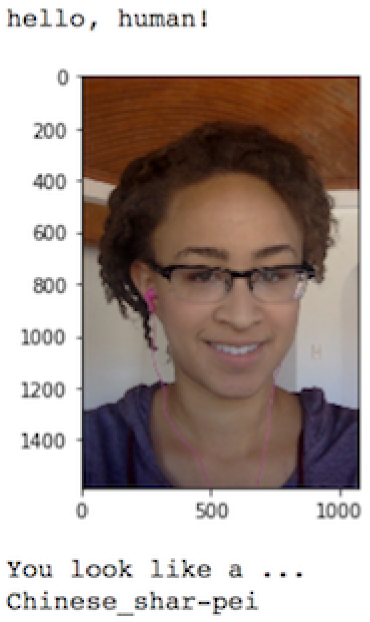
Step 6: Write your Algorithm

Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide output that indicates an error.

You are welcome to write your own functions for detecting humans and dogs in images, but feel free to use the `face_detector` and `dog_detector` functions developed above. You are **required** to use your CNN from Step 5 to predict dog breed.

Some sample output for our algorithm is provided below, but feel free to design your own user experience!



(IMPLEMENTATION) Write your Algorithm

Get image of Dog by breed name

In [53]:
```
def get_dog_image_file_by_breed(breedname):
    all_dog_files=[dfile for dfile in test_files]
    dog_images_file_list=[dog for dog in all_dog_files if breedname in dog]
    if len(dog_images_file_list)>0:
        return dog_images_file_list[0]
    return None
```

In [57]:
```
def print_two_images(orig_file,orig_label,title):
    fig = plt.figure(figsize=(5,5))
#     plt.title(title,loc='left')
#     fig.axes.clear
    fig.suptitle(title, fontsize=15)
    ax = fig.add_subplot(1, 2, 1, xticks=[], yticks=[])
    orig_image = cv2.imread(orig_file)
    orig_image = cv2.cvtColor(orig_image, cv2.COLOR_BGR2RGB)
    plt.xlabel(orig_label,size=12)
    ax.imshow(orig_image)

    ax = fig.add_subplot(1, 2, 2, xticks=[], yticks=[])
    breed=my_ResNet50_predict_breed(orig_file)
    if not breed:
        print("error getting breed name")
        return None
    breed_img=cv2.imread(get_dog_image_file_by_breed(breed))
    breed_img = cv2.cvtColor(breed_img, cv2.COLOR_BGR2RGB)
    plt.xlabel('you look like a '+breed,size=12)
    ax.imshow(breed_img)

    plt.subplots_adjust(left=0.29,right=1.89)
    plt.gcf().canvas.draw()
    return True

# print_two_images(file6,"human:atkinson")
```

In [68]:
```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.

def human_or_dog(orig_file,orig_label):
    is_human=face_detector2(orig_file)
    is_dog=dog_detector(orig_file)
    if is_human and not is_dog:
        title="Hello "+orig_label+" your a human."
    if is_dog and not is_human:
        title="Hello "+orig_label+" your a dog."
    if is_dog and is_human:
        title="Hello "+orig_label+" your a both a dog and a human."
    if not is_dog and not is_human:
        title="Hello "+orig_label+" your neither a dog or a human."
    print_two_images(orig_file,orig_label,title)
```

Step 7: Test Your Algorithm

In this section, you will take your new algorithm for a spin! What kind of dog does the algorithm think that **you** look like? If you have a dog, does it predict your dog's breed accurately? If you have a cat, does it mistakenly think that your cat is a dog?

(IMPLEMENTATION) Test Your Algorithm on Sample Images!

Test your algorithm at least six images on your computer. Feel free to use any images you like. Use at least two human and two dog images.

Question 6: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of improvement for your algorithm.

Answer:

The output is as expected, The CNN performed very well, and the ultimate test was that my daughter was very excited to see a dog breed that looked like her.

The output of the custom CNN dog_detector was incorrectly identifying one human image as not human, which is couter-intuitive, because it has all around higher test accuracy.

There are many areas for improvement, most of them would involve more advanced CNN construction and training functions.

Improvement #1, Creating a mini Resnet50 model.

```
The Resnet50 pretrained model, uses a lot of unrelated images, and has hundreds of
 Convolutional layers, including shortchtting.  It would be very useful to create a
n advanced CNN with a model similar to the resnet50 mode., using the keras api mode
l, that isnt as large, but incorporates shortcutting, batch normalization, and has
 far fewer layers, which could be trained only on the dog images datasets.
```

Improvement #2, Training improvements.

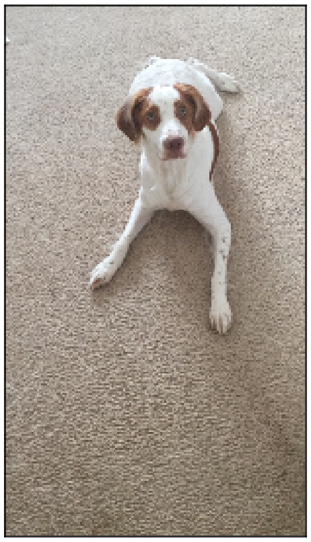
```
Using model saving, early stopping and learning rate reduction worked well for this
 project, but can be improved.  Specifically, when early stopping is hit, it alread
y proceeds a number of epochs past the actual optimal point, even when model saving
 is employed.  It would be beneficial to create a custom training callback that sav
es every training iteration, and monitor when the validation loss starts to deviate
 from the training loss.  When the deviation occurs, it makes sense to then revert
 several iterations before the deviation, retry the training or/and then simultaneo
usly lower the learning rate.
```

Improvement #3, Implement Grid search.

```
Using grid search would allow many more models as well as parameters to be tested t
o find the ultimate optimal solution.  This would best be done using an easy GPU de
ployment environment, such as Floydhub or AWS, and not with an IPython notebook.  T
his would allow many models to be tested, and dramatically incraese the ability to
 find the most accurate models and parameters, in the least amount of time.
```

In [69]:
```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.
human_or_dog('./our_brittany.jpg',"brittany")
human_or_dog('./myimages/atkinson.jpg',"atkinson")
human_or_dog('./myimages/lab.jpg',"labrador")
human_or_dog('./myimages/cocker_spanial.jpg',"cocker-spanial")
human_or_dog('./myimages/bulldog.jpg',"bulldog")
human_or_dog('./myimages/chihuahua.jpg',"chihuahua")
human_or_dog('./myimages/me.jpg',"andy")
human_or_dog('./myimages/sophia.jpg',"sophia")
human_or_dog('./myimages/wife.jpg',"victoria")
human_or_dog('./myimages/mom.jpg',"mom")
```

```
Hello brittany your a dog.
```


brittany

you look like a Brittany

```
Hello atkinson your a human.
```

atkinson

you look like a Beagle

```
Hello labrador your a dog.
```

labrador

you look like a Labrador_retriever

```
Hello cocker-spanial your a dog.
```

cocker-spanial

you look like a English_cocker_spaniel

```
Hello bulldog your a dog.
```

bulldog

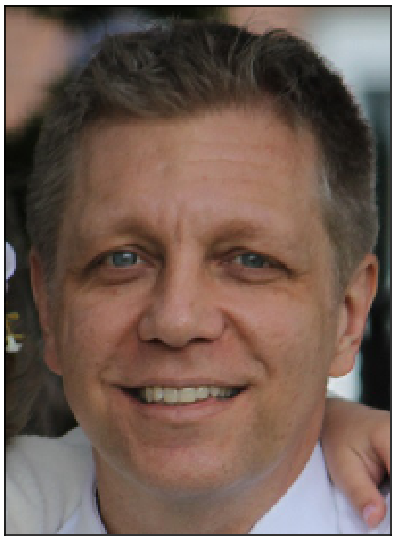
you look like a Bulldog

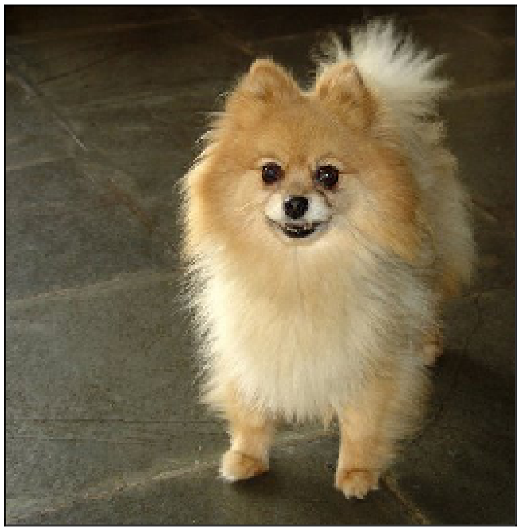
```
Hello chihuahua your a dog.
```

chihuahua

you look like a Chihuahua

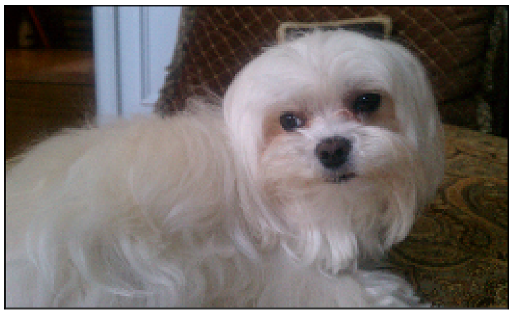
```
Hello andy your a human.
```


andy


you look like a Pomeranian

```
Hello sophia your a human.
```

sophia


you look like a Maltese

Hello victoria your a human.

victoria

you look like a Lowchen

Hello mom your a human.

mom

you look like a Havanese