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!).

Sample Dog Output

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 [2]:
```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from glob import glob
import random
import cv2
import matplotlib.pyplot as plt
from keras.preprocessing import image
from tqdm import tqdm
from keras.applications.resnet50 import preprocess_input, decode_predictions
from PIL import ImageFile
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
from extract_bottleneck_features import *
import cv2
import matplotlib.pyplot as plt
```

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('/data/dog_images/train')
valid_files, valid_targets = load_dataset('/data/dog_images/valid')
test_files, test_targets = load_dataset('/data/dog_images/test')

# load list of dog names
dog_names = [item[20:-1] for item in sorted(glob("/data/dog_images/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))
print( train_targets)
```

```
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.
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
```

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 [8]:
```
import random
random.seed(8675309)

# load filenames in shuffled human dataset
human_files = np.array(glob("/data/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.
```

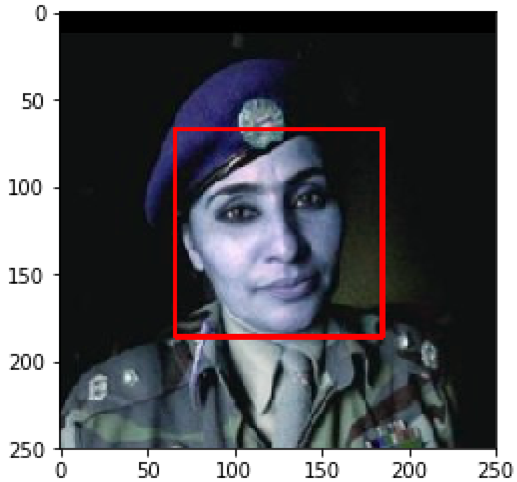
Step 1: Detect Humans

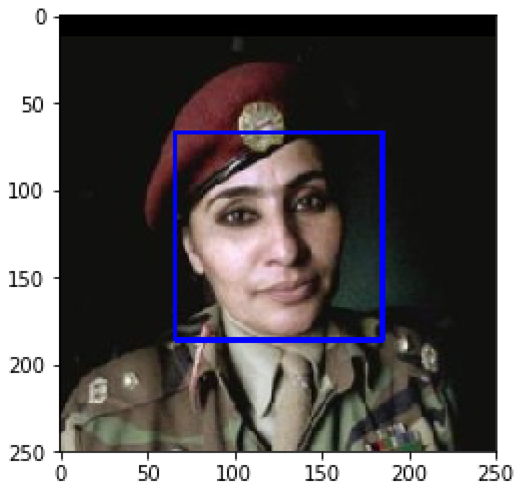
We use OpenCV's implementation of [Haar feature-based cascade classifiers (http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html)](http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html) to detect human faces in images. OpenCV provides many pre-trained face detectors, stored as XML files on [github (https://github.com/opencv/opencv/tree/master/data/haarcascades)](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 [9]:
```
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(img)
plt.show()
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 [10]:
```
# 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
```

(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:

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

h_face = 0
d_face = 0
for i in range(100):
    isFace = face_detector(human_files_short[i])
    isDog = face_detector(dog_files_short[i])
    if isFace:
        h_face = h_face +1
    if isDog:
        d_face = d_face +1
print(" percentage of the first 100 images in human_files have a detected huma
n face is {}%".format(h_face))

print("percentage of the first 100 images in dog_files have a detected human f
ace is  {}%".format(d_face))
## TODO: Test the performance of the face_detector algorithm
## on the images in human_files_short and dog_files_short.
```

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:

We suggest the face detector from OpenCV as a potential way to detect human images in your algorithm, but you are free to explore other approaches, especially approaches that make use of deep learning :). Please use the code cell below to design and test your own face detection algorithm. If you decide to pursue this *optional* task, report performance on each of the datasets.

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.
__Answer:__
I think the image needs to have certain standard of quality. If the face is co
vered mostly or if the face is completey back to camera or partilly and it si
not cleare
we cant expect alghorithm to detect face properly. I dont expect alghorithm de
tect face with poor light and quality, but if the
face is upside down or is to side or any other orientationn I would expect the
alghorthm detects the face .Here as we anyalized 10% of the dogs are
detected as human face for solving this might need to consider more fetures in
addition to (eyes, mouth, nose, etc
somthing like head shape or cheek
```

```
  File "<ipython-input-7-1d6d8344a97f>", line 5
    I think the image needs to have certain standard of quality. If the face
 is covered mostly or if the face is completey back to camera or partilly and
it si not cleare
          ^
SyntaxError: invalid syntax
```

Step 2: Detect Dogs

In this section, we use a pre-trained [ResNet-50 (http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006)](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/)](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)](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 [18]:
```
from keras.applications.resnet50 import ResNet50

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

```
Downloading data from https://github.com/fchollet/deep-learning-models/releas
es/download/v0.2/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102858752/102853048 [==============================] - 2s 0us/step
```

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 [16]:
```
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)](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)](https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a).

In [14]:
```
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)](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 [12]:
```
### 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:

In [ ]:
```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.
h_face = 0
d_face = 0
for i in range(100):
    isFace = dog_detector(human_files_short[i])
    isDog = dog_detector(dog_files_short[i])
    if isFace:
        h_face = h_face +1
    if isDog:
        d_face = d_face +1
print(" percentage of the first 100 images in human_files have a detected huma
n face is {}%".format(h_face))

print("percentage of the first 100 images in dog_files have a detected human f
ace is  {}%".format(d_face))
## TODO: Test the performance of the face_detector algorithm
## on the images in human_files_short and dog_files_short.
```

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 [8]:
```
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 [01:29<00:00, 75.06it/s]
100%|██████████| 835/835 [00:09<00:00, 83.62it/s]
100%|██████████| 836/836 [00:09<00:00, 86.04it/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:

Sample CNN

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:

In [10]:
```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

model = Sequential()

### TODO: Define your architecture.
model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
                 input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'
))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'
))
model.add(MaxPooling2D(pool_size=2))
model.add(GlobalAveragePooling2D())
model.add(Dense(133, activation='relu'))


# summarize the model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 224, 224, 16)      208       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 112, 112, 32)      2080      
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 64)        8256      
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 28, 28, 64)        0         
_________________________________________________________________
global_average_pooling2d_2 ( (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 133)               8645      
=================================================================
Total params: 19,189
Trainable params: 19,189
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['a
ccuracy'])
```

Compile the Model

(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)](https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html), but this is not a requirement.

In [15]:
```
from keras.callbacks import ModelCheckpoint

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

epochs = 20

### Do NOT modify the code below this line.

checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.from_scratc
h.hdf5',
                               verbose=1, save_best_only=True)

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 1/20
6660/6680 [============================>.] - ETA: 0s - loss: 11.3008 - acc:
0.0081Epoch 00001: val_loss improved from inf to 12.30782, saving model to sa
ved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 25s 4ms/step - loss: 11.3064 - a
cc: 0.0081 - val_loss: 12.3078 - val_acc: 0.0108
Epoch 2/20
6660/6680 [============================>.] - ETA: 0s - loss: 12.9217 - acc:
0.0072Epoch 00002: val_loss improved from 12.30782 to 12.29474, saving model
to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 22s 3ms/step - loss: 12.9239 - a
cc: 0.0072 - val_loss: 12.2947 - val_acc: 0.0096
Epoch 3/20
6660/6680 [============================>.] - ETA: 0s - loss: 12.3246 - acc:
0.0105Epoch 00003: val_loss improved from 12.29474 to 11.73374, saving model
to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 22s 3ms/step - loss: 12.3198 - a
cc: 0.0105 - val_loss: 11.7337 - val_acc: 0.0096
Epoch 4/20
6660/6680 [============================>.] - ETA: 0s - loss: 11.6408 - acc:
0.0096Epoch 00004: val_loss did not improve
6680/6680 [==============================] - 21s 3ms/step - loss: 11.6415 - a
cc: 0.0096 - val_loss: 11.9196 - val_acc: 0.0108
Epoch 5/20
6660/6680 [============================>.] - ETA: 0s - loss: 13.0199 - acc:
0.0090Epoch 00005: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 13.0292 - a
cc: 0.0090 - val_loss: 14.5551 - val_acc: 0.0096
Epoch 6/20
6660/6680 [============================>.] - ETA: 0s - loss: 13.7346 - acc:
0.0114Epoch 00006: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 13.7320 - a
cc: 0.0114 - val_loss: 13.4172 - val_acc: 0.0084
Epoch 7/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.1382 - acc:
0.0090Epoch 00007: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 14.1420 - a
cc: 0.0090 - val_loss: 14.9622 - val_acc: 0.0096
Epoch 8/20
6660/6680 [============================>.] - ETA: 0s - loss: 15.6113 - acc:
0.0099Epoch 00008: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 15.6105 - a
cc: 0.0100 - val_loss: 15.8229 - val_acc: 0.0096
Epoch 9/20
6660/6680 [============================>.] - ETA: 0s - loss: 15.7919 - acc:
0.0078Epoch 00009: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 15.7929 - a
cc: 0.0078 - val_loss: 15.7493 - val_acc: 0.0096
Epoch 10/20
6660/6680 [============================>.] - ETA: 0s - loss: 15.9046 - acc:
0.0071Epoch 00010: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 15.9052 - a
cc: 0.0070 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 11/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00011: val_loss did not improve
```

```
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 12/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00012: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 13/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0237 - acc:
0.0059Epoch 00013: val_loss did not improve
6680/6680 [==============================] - 23s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 14/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00014: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 15/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00015: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 16/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00016: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 17/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00017: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 18/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00018: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 19/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00019: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
Epoch 20/20
6660/6680 [============================>.] - ETA: 0s - loss: 16.0213 - acc:
0.0060Epoch 00020: val_loss did not improve
6680/6680 [==============================] - 22s 3ms/step - loss: 16.0216 - a
cc: 0.0060 - val_loss: 16.0216 - val_acc: 0.0060
```

Out[15]:

```
<keras.callbacks.History at 0x7f03358df2b0>
```

Load the Model with the Best Validation Loss

In [16]:

```python
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 [17]:
```
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, 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('Test accuracy: %.4f%%' % test_accuracy)
```

```
Test accuracy: 0.9569%
```

In [ ]:
```
###Lenet
```

In [ ]:
```
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('/data/dog_images/train')
valid_files, valid_targets = load_dataset('/data/dog_images/valid')
test_files, test_targets = load_dataset('/data/dog_images/test')

# load list of dog names
dog_names = [item[20:-1] for item in sorted(glob("/data/dog_images/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))
#print( train_targets)
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.
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)
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 [01:27<00:00, 76.69it/s]
100%|██████████| 835/835 [00:09<00:00, 85.24it/s]
100%|██████████| 836/836 [00:09<00:00, 85.82it/s]
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D,MaxPool2
D,AveragePooling2D
from keras.layers import Dropout, Flatten, Dense, Input, concatenate
from keras.models import Sequential,Model
from keras.layers.core import Activation
from keras.layers.core import Flatten
from pyimagesearch.cnn.networks.lenet import LeNet
from sklearn.model_selection import train_test_split
from keras.datasets import mnist
from keras.optimizers import SGD
from keras.utils import np_utils
from keras import backend as K
import numpy as np
import argparse
import cv2
model = Sequential()
Using TensorFlow backend.
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-b69f81eb32f8> in <module>()
      4 from keras.layers.core import Activation
      5 from keras.layers.core import Flatten
----> 6 from pyimagesearch.cnn.networks.lenet import LeNet
      7 from sklearn.model_selection import train_test_split
      8 from keras.datasets import mnist

ModuleNotFoundError: No module named 'pyimagesearch'

# define the first set of CONV => ACTIVATION => POOL layers
model.add(Conv2D(20, 5, padding="same",input_shape=(224,224,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
# define the second set of CONV => ACTIVATION => POOL layers
model.add(Conv2D(50, 5, padding="same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
# define the first FC => ACTIVATION layers
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))

# define the second FC layer
model.add(Dense(133))

# lastly, define the soft-max classifier
model.add(Activation("softmax"))

model.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accurac
y"])
model.fit(train_tensors, train_targets,validation_data =(valid_tensors,valid_t
```

```
argets ), batch_size=128, epochs=20,verbose=1)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-dd58505724a0> in <module>()
----> 1 model.compile(loss="categorical_crossentropy", optimizer=opt,metrics=[
"accuracy"])
      2 model.fit(train_tensors, train_targets,validation_data =(valid_tensors
,valid_targets ), batch_size=128, epochs=20,verbose=1)

NameError: name 'opt' is not defined
```

```
# show the accuracy on the testing set
print("[INFO] evaluating...")
(loss, accuracy) = model.evaluate(testData, testLabels,
batch_size=128, verbose=1)
print("[INFO] accuracy: {:.2f}%".format(accuracy * 100))
```

```
model.summary()
```

```
______________________________________________________________________________
____________________
Layer (type)                    Output Shape         Param #     Connected to
==============================================================================
====================
input_1 (InputLayer)            (None, 224, 224, 3)  0
______________________________________________________________________________
____________________
conv_1_7x7/2 (Conv2D)           (None, 112, 112, 64) 9472        input_1[0][0]
______________________________________________________________________________
____________________
max_pool_1_3x3/2 (MaxPooling2D) (None, 56, 56, 64)   0           conv_1_7x7/2[
0][0]
______________________________________________________________________________
____________________
conv_2a_3x3/1 (Conv2D)          (None, 56, 56, 64)   4160        max_pool_1_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv_2b_3x3/1 (Conv2D)          (None, 56, 56, 192)  110784      conv_2a_3x3/1
[0][0]
______________________________________________________________________________
____________________
max_pool_2_3x3/2 (MaxPooling2D) (None, 28, 28, 192)  0           conv_2b_3x3/1
[0][0]
______________________________________________________________________________
____________________
conv2d_2 (Conv2D)               (None, 28, 28, 96)   18528       max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_4 (Conv2D)               (None, 28, 28, 16)   3088        max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
____________________
max_pooling2d_1 (MaxPooling2D)  (None, 28, 28, 192)  0           max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
```

```
____________________
conv2d_1 (Conv2D)               (None, 28, 28, 64)   12352       max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_3 (Conv2D)               (None, 28, 28, 128)  110720      conv2d_2[0][0
]
______________________________________________________________________________
____________________
conv2d_5 (Conv2D)               (None, 28, 28, 32)   12832       conv2d_4[0][0
]
______________________________________________________________________________
____________________
conv2d_6 (Conv2D)               (None, 28, 28, 32)   6176        max_pooling2d
_1[0][0]
______________________________________________________________________________
____________________
inception_3a (Concatenate)      (None, 28, 28, 256)  0           conv2d_1[0][0
]
                                                                 conv2d_3[0][0
]
                                                                 conv2d_5[0][0
]
                                                                 conv2d_6[0][0
]
______________________________________________________________________________
____________________
conv2d_8 (Conv2D)               (None, 28, 28, 128)  32896       inception_3a[
0][0]
______________________________________________________________________________
____________________
conv2d_10 (Conv2D)              (None, 28, 28, 32)   8224        inception_3a[
0][0]
______________________________________________________________________________
____________________
max_pooling2d_2 (MaxPooling2D)  (None, 28, 28, 256)  0           inception_3a[
0][0]
______________________________________________________________________________
____________________
conv2d_7 (Conv2D)               (None, 28, 28, 128)  32896       inception_3a[
0][0]
______________________________________________________________________________
____________________
conv2d_9 (Conv2D)               (None, 28, 28, 192)  221376      conv2d_8[0][0
]
______________________________________________________________________________
____________________
conv2d_11 (Conv2D)              (None, 28, 28, 96)   76896       conv2d_10[0][
0]
______________________________________________________________________________
____________________
conv2d_12 (Conv2D)              (None, 28, 28, 64)   16448       max_pooling2d
_2[0][0]
______________________________________________________________________________
____________________
inception_3b (Concatenate)      (None, 28, 28, 480)  0           conv2d_7[0][0
]
```

```
                                                                conv2d_9[0][0
]
                                                                conv2d_11[0][
0]
                                                                conv2d_12[0][
0]
______________________________________________________________________________
____________________
max_pool_3_3x3/2 (MaxPooling2D) (None, 14, 14, 480)  0          inception_3b[
0][0]
______________________________________________________________________________
____________________
conv2d_14 (Conv2D)              (None, 14, 14, 96)   46176      max_pool_3_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_16 (Conv2D)              (None, 14, 14, 16)   7696       max_pool_3_3x
3/2[0][0]
______________________________________________________________________________
____________________
max_pooling2d_3 (MaxPooling2D)  (None, 14, 14, 480)  0          max_pool_3_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_13 (Conv2D)              (None, 14, 14, 192)  92352      max_pool_3_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_15 (Conv2D)              (None, 14, 14, 208)  179920     conv2d_14[0][
0]
______________________________________________________________________________
____________________
conv2d_17 (Conv2D)              (None, 14, 14, 48)   19248      conv2d_16[0][
0]
______________________________________________________________________________
____________________
conv2d_18 (Conv2D)              (None, 14, 14, 64)   30784      max_pooling2d
_3[0][0]
______________________________________________________________________________
____________________
inception_4a (Concatenate)      (None, 14, 14, 512)  0          conv2d_13[0][
0]
                                                                conv2d_15[0][
0]
                                                                conv2d_17[0][
0]
                                                                conv2d_18[0][
0]
______________________________________________________________________________
____________________
conv2d_21 (Conv2D)              (None, 14, 14, 112)  57456      inception_4a[
0][0]
______________________________________________________________________________
____________________
conv2d_23 (Conv2D)              (None, 14, 14, 24)   12312      inception_4a[
0][0]
______________________________________________________________________________
```

```
max_pooling2d_4 (MaxPooling2D)  (None, 14, 14, 512)  0           inception_4a[
0][0]
__________________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, 14, 14, 160)  82080       inception_4a[
0][0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 14, 14, 224)  226016      conv2d_21[0][
0]
__________________________________________________________________________________________________
conv2d_24 (Conv2D)              (None, 14, 14, 64)   38464       conv2d_23[0][
0]
__________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 14, 14, 64)   32832       max_pooling2d
_4[0][0]
__________________________________________________________________________________________________
inception_4b (Concatenate)      (None, 14, 14, 512)  0           conv2d_20[0][
0]
                                                                 conv2d_22[0][
0]
                                                                 conv2d_24[0][
0]
                                                                 conv2d_25[0][
0]
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 14, 14, 128)  65664       inception_4b[
0][0]
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 14, 14, 24)   12312       inception_4b[
0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 14, 14, 512)  0           inception_4b[
0][0]
__________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 14, 14, 128)  65664       inception_4b[
0][0]
__________________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 14, 14, 256)  295168      conv2d_27[0][
0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 14, 14, 64)   38464       conv2d_29[0][
0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 14, 14, 64)   32832       max_pooling2d
_5[0][0]
```

```
________________________________________________________________________________
____________________
inception_4c (Concatenate)      (None, 14, 14, 512)  0           conv2d_26[0][
0]
                                                                 conv2d_28[0][
0]
                                                                 conv2d_30[0][
0]
                                                                 conv2d_31[0][
0]
________________________________________________________________________________
____________________
conv2d_33 (Conv2D)              (None, 14, 14, 144)  73872       inception_4c[
0][0]
________________________________________________________________________________
____________________
conv2d_35 (Conv2D)              (None, 14, 14, 32)   16416       inception_4c[
0][0]
________________________________________________________________________________
____________________
max_pooling2d_6 (MaxPooling2D)  (None, 14, 14, 512)  0           inception_4c[
0][0]
________________________________________________________________________________
____________________
conv2d_32 (Conv2D)              (None, 14, 14, 112)  57456       inception_4c[
0][0]
________________________________________________________________________________
____________________
conv2d_34 (Conv2D)              (None, 14, 14, 288)  373536      conv2d_33[0][
0]
________________________________________________________________________________
____________________
conv2d_36 (Conv2D)              (None, 14, 14, 64)   51264       conv2d_35[0][
0]
________________________________________________________________________________
____________________
conv2d_37 (Conv2D)              (None, 14, 14, 64)   32832       max_pooling2d
_6[0][0]
________________________________________________________________________________
____________________
inception_4d (Concatenate)      (None, 14, 14, 528)  0           conv2d_32[0][
0]
                                                                 conv2d_34[0][
0]
                                                                 conv2d_36[0][
0]
                                                                 conv2d_37[0][
0]
________________________________________________________________________________
____________________
conv2d_40 (Conv2D)              (None, 14, 14, 160)  84640       inception_4d[
0][0]
________________________________________________________________________________
____________________
conv2d_42 (Conv2D)              (None, 14, 14, 32)   16928       inception_4d[
0][0]
________________________________________________________________________________
```

```
____________________________
max_pooling2d_7 (MaxPooling2D)  (None, 14, 14, 528)  0           inception_4d[
0][0]
__________________________________________________________________________________________________
____________________________
conv2d_39 (Conv2D)              (None, 14, 14, 256)  135424      inception_4d[
0][0]
__________________________________________________________________________________________________
____________________________
conv2d_41 (Conv2D)              (None, 14, 14, 320)  461120      conv2d_40[0][
0]
__________________________________________________________________________________________________
____________________________
conv2d_43 (Conv2D)              (None, 14, 14, 128)  102528      conv2d_42[0][
0]
__________________________________________________________________________________________________
____________________________
conv2d_44 (Conv2D)              (None, 14, 14, 128)  67712       max_pooling2d
_7[0][0]
__________________________________________________________________________________________________
____________________________
inception_4e (Concatenate)      (None, 14, 14, 832)  0           conv2d_39[0][
0]
                                                                 conv2d_41[0][
0]
                                                                 conv2d_43[0][
0]
                                                                 conv2d_44[0][
0]
__________________________________________________________________________________________________
____________________________
max_pool_4_3x3/2 (MaxPooling2D) (None, 7, 7, 832)    0           inception_4e[
0][0]
__________________________________________________________________________________________________
____________________________
conv2d_46 (Conv2D)              (None, 7, 7, 160)    133280      max_pool_4_3x
3/2[0][0]
__________________________________________________________________________________________________
____________________________
conv2d_48 (Conv2D)              (None, 7, 7, 32)     26656       max_pool_4_3x
3/2[0][0]
__________________________________________________________________________________________________
____________________________
max_pooling2d_8 (MaxPooling2D)  (None, 7, 7, 832)    0           max_pool_4_3x
3/2[0][0]
__________________________________________________________________________________________________
____________________________
conv2d_45 (Conv2D)              (None, 7, 7, 256)    213248      max_pool_4_3x
3/2[0][0]
__________________________________________________________________________________________________
____________________________
conv2d_47 (Conv2D)              (None, 7, 7, 320)    461120      conv2d_46[0][
0]
__________________________________________________________________________________________________
____________________________
conv2d_49 (Conv2D)              (None, 7, 7, 128)    102528      conv2d_48[0][
0]
```

```
____________________________________________________________________________________________
__________________________
conv2d_50 (Conv2D)              (None, 7, 7, 128)    106624      max_pooling2d
_8[0][0]
____________________________________________________________________________________________
__________________________
inception_5a (Concatenate)      (None, 7, 7, 832)    0           conv2d_45[0][
0]
                                                                 conv2d_47[0][
0]
                                                                 conv2d_49[0][
0]
                                                                 conv2d_50[0][
0]
____________________________________________________________________________________________
__________________________
conv2d_52 (Conv2D)              (None, 7, 7, 192)    159936      inception_5a[
0][0]
____________________________________________________________________________________________
__________________________
conv2d_54 (Conv2D)              (None, 7, 7, 48)     39984       inception_5a[
0][0]
____________________________________________________________________________________________
__________________________
max_pooling2d_9 (MaxPooling2D)  (None, 7, 7, 832)    0           inception_5a[
0][0]
____________________________________________________________________________________________
__________________________
average_pooling2d_1 (AveragePoo (None, 4, 4, 512)    0           inception_4a[
0][0]
____________________________________________________________________________________________
__________________________
average_pooling2d_2 (AveragePoo (None, 4, 4, 528)    0           inception_4d[
0][0]
____________________________________________________________________________________________
__________________________
conv2d_51 (Conv2D)              (None, 7, 7, 384)    319872      inception_5a[
0][0]
____________________________________________________________________________________________
__________________________
conv2d_53 (Conv2D)              (None, 7, 7, 384)    663936      conv2d_52[0][
0]
____________________________________________________________________________________________
__________________________
conv2d_55 (Conv2D)              (None, 7, 7, 128)    153728      conv2d_54[0][
0]
____________________________________________________________________________________________
__________________________
conv2d_56 (Conv2D)              (None, 7, 7, 128)    106624      max_pooling2d
_9[0][0]
____________________________________________________________________________________________
__________________________
conv2d_19 (Conv2D)              (None, 4, 4, 128)    65664       average_pooli
ng2d_1[0][0]
____________________________________________________________________________________________
__________________________
conv2d_38 (Conv2D)              (None, 4, 4, 128)    67712       average_pooli
```

```
ng2d_2[0][0]
__________________________________________________________________________________________________
inception_5b (Concatenate)      (None, 7, 7, 1024)   0           conv2d_51[0][
0]
                                                                 conv2d_53[0][
0]
                                                                 conv2d_55[0][
0]
                                                                 conv2d_56[0][
0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 2048)         0           conv2d_19[0][
0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 2048)         0           conv2d_38[0][
0]
__________________________________________________________________________________________________
avg_pool_5_3x3/1 (GlobalAverage (None, 1024)         0           inception_5b[
0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1024)         2098176     flatten_1[0][
0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1024)         2098176     flatten_2[0][
0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 1024)         0           avg_pool_5_3x
3/1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 1024)         0           dense_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 1024)         0           dense_2[0][0]
__________________________________________________________________________________________________
output (Dense)                  (None, 133)          136325      dropout_3[0][
0]
__________________________________________________________________________________________________
auxilliary_output_1 (Dense)     (None, 133)          136325      dropout_1[0][
0]
__________________________________________________________________________________________________
auxilliary_output_2 (Dense)     (None, 133)          136325      dropout_2[0][
0]
==================================================================================================
Total params: 10,712,255
Trainable params: 10,712,255
```

```
Non-trainable params: 0
________________________________________________________________________________
________________
import cv2
import numpy as np
from keras.datasets import cifar10
from keras import backend as K
from keras.utils import np_utils

import math
from keras.optimizers import SGD
from keras.callbacks import LearningRateScheduler
epochs = 25
initial_lrate = 0.01

def decay(epoch, steps=100):
    initial_lrate = 0.01
    drop = 0.96
    epochs_drop = 8
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lrate

sgd = SGD(lr=initial_lrate, momentum=0.9, nesterov=False)

lr_sc = LearningRateScheduler(decay)

#model.compile(loss=['categorical_crossentropy', 'categorical_crossentropy',
 'categorical_crossentropy'], loss_weights=[1, 0.3, 0.3], optimizer=sgd, metri
cs=['accuracy'])
#model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=
['accuracy'])
model.compile(loss=['categorical_crossentropy', 'categorical_crossentropy', 'c
ategorical_crossentropy'], loss_weights=[1, 0.3, 0.3], optimizer=sgd, metrics=
['accuracy'])

history = model.fit(train_tensors, [train_targets, train_targets, train_target
s], validation_data=(valid_tensors, [valid_targets, valid_targets, valid_targe
ts]), epochs=epochs, batch_size=256, callbacks=[lr_sc])
Train on 6680 samples, validate on 835 samples
Epoch 1/25
6680/6680 [==============================] - 73s 11ms/step - loss: 7.9343 - ou
tput_loss: 4.9714 - auxilliary_output_1_loss: 4.9415 - auxilliary_output_2_los
s: 4.9346 - output_acc: 0.0091 - auxilliary_output_1_acc: 0.0076 - auxilliary_
output_2_acc: 0.0067 - val_loss: 7.8149 - val_output_loss: 4.8810 - val_auxill
iary_output_1_loss: 4.8901 - val_auxilliary_output_2_loss: 4.8897 - val_output
_acc: 0.0096 - val_auxilliary_output_1_acc: 0.0048 - val_auxilliary_output_2_a
cc: 0.0096
Epoch 2/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8485 - out
put_loss: 4.9136 - auxilliary_output_1_loss: 4.8926 - auxilliary_output_2_loss
: 4.8907 - output_acc: 0.0075 - auxilliary_output_1_acc: 0.0073 - auxilliary_o
utput_2_acc: 0.0081 - val_loss: 7.8110 - val_output_loss: 4.8783 - val_auxilli
ary_output_1_loss: 4.8888 - val_auxilliary_output_2_loss: 4.8869 - val_output_
acc: 0.0132 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 3/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8267 - out
```

```
put_loss: 4.8932 - auxilliary_output_1_loss: 4.8895 - auxilliary_output_2_loss
: 4.8888 - output_acc: 0.0087 - auxilliary_output_1_acc: 0.0069 - auxilliary_o
utput_2_acc: 0.0079 - val_loss: 7.8064 - val_output_loss: 4.8742 - val_auxilli
ary_output_1_loss: 4.8883 - val_auxilliary_output_2_loss: 4.8857 - val_output_
acc: 0.0096 - val_auxilliary_output_1_acc: 0.0072 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 4/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8208 - out
put_loss: 4.8870 - auxilliary_output_1_loss: 4.8902 - auxilliary_output_2_loss
: 4.8892 - output_acc: 0.0090 - auxilliary_output_1_acc: 0.0096 - auxilliary_o
utput_2_acc: 0.0100 - val_loss: 7.8030 - val_output_loss: 4.8710 - val_auxilli
ary_output_1_loss: 4.8875 - val_auxilliary_output_2_loss: 4.8858 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 5/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8173 - out
put_loss: 4.8845 - auxilliary_output_1_loss: 4.8881 - auxilliary_output_2_loss
: 4.8879 - output_acc: 0.0106 - auxilliary_output_1_acc: 0.0079 - auxilliary_o
utput_2_acc: 0.0079 - val_loss: 7.8029 - val_output_loss: 4.8709 - val_auxilli
ary_output_1_loss: 4.8870 - val_auxilliary_output_2_loss: 4.8862 - val_output_
acc: 0.0084 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 6/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8115 - out
put_loss: 4.8790 - auxilliary_output_1_loss: 4.8882 - auxilliary_output_2_loss
: 4.8868 - output_acc: 0.0078 - auxilliary_output_1_acc: 0.0091 - auxilliary_o
utput_2_acc: 0.0085 - val_loss: 7.8022 - val_output_loss: 4.8707 - val_auxilli
ary_output_1_loss: 4.8867 - val_auxilliary_output_2_loss: 4.8849 - val_output_
acc: 0.0096 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 7/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8102 - out
put_loss: 4.8784 - auxilliary_output_1_loss: 4.8867 - auxilliary_output_2_loss
: 4.8859 - output_acc: 0.0094 - auxilliary_output_1_acc: 0.0094 - auxilliary_o
utput_2_acc: 0.0096 - val_loss: 7.8016 - val_output_loss: 4.8709 - val_auxilli
ary_output_1_loss: 4.8856 - val_auxilliary_output_2_loss: 4.8835 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 8/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8068 - out
put_loss: 4.8757 - auxilliary_output_1_loss: 4.8844 - auxilliary_output_2_loss
: 4.8857 - output_acc: 0.0081 - auxilliary_output_1_acc: 0.0105 - auxilliary_o
utput_2_acc: 0.0093 - val_loss: 7.8010 - val_output_loss: 4.8710 - val_auxilli
ary_output_1_loss: 4.8837 - val_auxilliary_output_2_loss: 4.8830 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 9/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8086 - out
put_loss: 4.8780 - auxilliary_output_1_loss: 4.8851 - auxilliary_output_2_loss
: 4.8836 - output_acc: 0.0078 - auxilliary_output_1_acc: 0.0076 - auxilliary_o
utput_2_acc: 0.0103 - val_loss: 7.7991 - val_output_loss: 4.8698 - val_auxilli
ary_output_1_loss: 4.8831 - val_auxilliary_output_2_loss: 4.8813 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 10/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8039 - out
put_loss: 4.8739 - auxilliary_output_1_loss: 4.8824 - auxilliary_output_2_loss
```

```
: 4.8843 - output_acc: 0.0082 - auxilliary_output_1_acc: 0.0091 - auxilliary_o
utput_2_acc: 0.0096 - val_loss: 7.7987 - val_output_loss: 4.8696 - val_auxilli
ary_output_1_loss: 4.8816 - val_auxilliary_output_2_loss: 4.8820 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0072 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 11/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8014 - out
put_loss: 4.8730 - auxilliary_output_1_loss: 4.8806 - auxilliary_output_2_loss
: 4.8806 - output_acc: 0.0079 - auxilliary_output_1_acc: 0.0085 - auxilliary_o
utput_2_acc: 0.0108 - val_loss: 7.7970 - val_output_loss: 4.8698 - val_auxilli
ary_output_1_loss: 4.8796 - val_auxilliary_output_2_loss: 4.8779 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 12/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8013 - out
put_loss: 4.8727 - auxilliary_output_1_loss: 4.8809 - auxilliary_output_2_loss
: 4.8809 - output_acc: 0.0097 - auxilliary_output_1_acc: 0.0099 - auxilliary_o
utput_2_acc: 0.0085 - val_loss: 7.7975 - val_output_loss: 4.8701 - val_auxilli
ary_output_1_loss: 4.8791 - val_auxilliary_output_2_loss: 4.8788 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0072 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 13/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.8008 - out
put_loss: 4.8727 - auxilliary_output_1_loss: 4.8805 - auxilliary_output_2_loss
: 4.8798 - output_acc: 0.0091 - auxilliary_output_1_acc: 0.0082 - auxilliary_o
utput_2_acc: 0.0111 - val_loss: 7.7968 - val_output_loss: 4.8701 - val_auxilli
ary_output_1_loss: 4.8781 - val_auxilliary_output_2_loss: 4.8775 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 14/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7999 - out
put_loss: 4.8727 - auxilliary_output_1_loss: 4.8783 - auxilliary_output_2_loss
: 4.8788 - output_acc: 0.0105 - auxilliary_output_1_acc: 0.0123 - auxilliary_o
utput_2_acc: 0.0103 - val_loss: 7.7963 - val_output_loss: 4.8705 - val_auxilli
ary_output_1_loss: 4.8761 - val_auxilliary_output_2_loss: 4.8765 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0048 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 15/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7983 - out
put_loss: 4.8716 - auxilliary_output_1_loss: 4.8781 - auxilliary_output_2_loss
: 4.8775 - output_acc: 0.0118 - auxilliary_output_1_acc: 0.0099 - auxilliary_o
utput_2_acc: 0.0111 - val_loss: 7.7954 - val_output_loss: 4.8698 - val_auxilli
ary_output_1_loss: 4.8768 - val_auxilliary_output_2_loss: 4.8753 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0084 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 16/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7970 - out
put_loss: 4.8703 - auxilliary_output_1_loss: 4.8768 - auxilliary_output_2_loss
: 4.8787 - output_acc: 0.0091 - auxilliary_output_1_acc: 0.0103 - auxilliary_o
utput_2_acc: 0.0073 - val_loss: 7.7944 - val_output_loss: 4.8695 - val_auxilli
ary_output_1_loss: 4.8745 - val_auxilliary_output_2_loss: 4.8754 - val_output_
acc: 0.0096 - val_auxilliary_output_1_acc: 0.0084 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 17/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7952 - out
put_loss: 4.8694 - auxilliary_output_1_loss: 4.8753 - auxilliary_output_2_loss
: 4.8776 - output_acc: 0.0123 - auxilliary_output_1_acc: 0.0100 - auxilliary_o
```

```
utput_2_acc: 0.0088 - val_loss: 7.7937 - val_output_loss: 4.8692 - val_auxilli
ary_output_1_loss: 4.8734 - val_auxilliary_output_2_loss: 4.8749 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0084 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 18/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7949 - out
put_loss: 4.8695 - auxilliary_output_1_loss: 4.8746 - auxilliary_output_2_loss
: 4.8768 - output_acc: 0.0082 - auxilliary_output_1_acc: 0.0112 - auxilliary_o
utput_2_acc: 0.0088 - val_loss: 7.7937 - val_output_loss: 4.8695 - val_auxilli
ary_output_1_loss: 4.8731 - val_auxilliary_output_2_loss: 4.8741 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 19/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7948 - out
put_loss: 4.8699 - auxilliary_output_1_loss: 4.8740 - auxilliary_output_2_loss
: 4.8757 - output_acc: 0.0097 - auxilliary_output_1_acc: 0.0090 - auxilliary_o
utput_2_acc: 0.0100 - val_loss: 7.7927 - val_output_loss: 4.8691 - val_auxilli
ary_output_1_loss: 4.8720 - val_auxilliary_output_2_loss: 4.8732 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 20/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7917 - out
put_loss: 4.8677 - auxilliary_output_1_loss: 4.8717 - auxilliary_output_2_loss
: 4.8749 - output_acc: 0.0099 - auxilliary_output_1_acc: 0.0109 - auxilliary_o
utput_2_acc: 0.0111 - val_loss: 7.7922 - val_output_loss: 4.8689 - val_auxilli
ary_output_1_loss: 4.8710 - val_auxilliary_output_2_loss: 4.8734 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0096 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 21/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7952 - out
put_loss: 4.8719 - auxilliary_output_1_loss: 4.8692 - auxilliary_output_2_loss
: 4.8753 - output_acc: 0.0114 - auxilliary_output_1_acc: 0.0112 - auxilliary_o
utput_2_acc: 0.0105 - val_loss: 7.7916 - val_output_loss: 4.8688 - val_auxilli
ary_output_1_loss: 4.8698 - val_auxilliary_output_2_loss: 4.8728 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 22/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7911 - out
put_loss: 4.8679 - auxilliary_output_1_loss: 4.8709 - auxilliary_output_2_loss
: 4.8732 - output_acc: 0.0096 - auxilliary_output_1_acc: 0.0126 - auxilliary_o
utput_2_acc: 0.0090 - val_loss: 7.7910 - val_output_loss: 4.8687 - val_auxilli
ary_output_1_loss: 4.8690 - val_auxilliary_output_2_loss: 4.8718 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0060 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 23/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7915 - out
put_loss: 4.8689 - auxilliary_output_1_loss: 4.8685 - auxilliary_output_2_loss
: 4.8734 - output_acc: 0.0106 - auxilliary_output_1_acc: 0.0114 - auxilliary_o
utput_2_acc: 0.0102 - val_loss: 7.7915 - val_output_loss: 4.8690 - val_auxilli
ary_output_1_loss: 4.8683 - val_auxilliary_output_2_loss: 4.8731 - val_output_
acc: 0.0108 - val_auxilliary_output_1_acc: 0.0120 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 24/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7925 - out
put_loss: 4.8701 - auxilliary_output_1_loss: 4.8686 - auxilliary_output_2_loss
: 4.8729 - output_acc: 0.0099 - auxilliary_output_1_acc: 0.0126 - auxilliary_o
utput_2_acc: 0.0124 - val_loss: 7.7903 - val_output_loss: 4.8690 - val_auxilli
```

```
ary_output_1_loss: 4.8671 - val_auxilliary_output_2_loss: 4.8706 - val_output_
acc: 0.0096 - val_auxilliary_output_1_acc: 0.0144 - val_auxilliary_output_2_ac
c: 0.0108
Epoch 25/25
6680/6680 [==============================] - 63s 9ms/step - loss: 7.7903 - out
put_loss: 4.8673 - auxilliary_output_1_loss: 4.8695 - auxilliary_output_2_loss
: 4.8736 - output_acc: 0.0123 - auxilliary_output_1_acc: 0.0094 - auxilliary_o
utput_2_acc: 0.0126 - val_loss: 7.7901 - val_output_loss: 4.8686 - val_auxilli
ary_output_1_loss: 4.8661 - val_auxilliary_output_2_loss: 4.8720 - val_output_
acc: 0.0120 - val_auxilliary_output_1_acc: 0.0084 - val_auxilliary_output_2_ac
c: 0.0108
#model.load_weights('saved_models/weights.best.from_scratch.hdf5')
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, 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('Test accuracy: %.4f%%' % test_accuracy)
Test accuracy: 0.0000%
```

In [ ]:

```
###LeNet 5
```

In [ ]:
```
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('/data/dog_images/train')
valid_files, valid_targets = load_dataset('/data/dog_images/valid')
test_files, test_targets = load_dataset('/data/dog_images/test')

# load list of dog names
dog_names = [item[20:-1] for item in sorted(glob("/data/dog_images/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))
#print( train_targets)
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.
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)
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 [01:25<00:00, 78.06it/s]
100%|██████████| 835/835 [00:09<00:00, 97.09it/s]
100%|██████████| 836/836 [00:09<00:00, 86.46it/s]
from keras.models import Sequential
from keras import models, layers
import keras
#Instantiate an empty model
model = Sequential()

# C1 Convolutional Layer
model.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation="tan
h", input_shape=(224,224,3), padding="same"))

# S2 Pooling Layer
model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padding="v
alid"))

# C3 Convolutional Layer
model.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation="ta
nh", padding="valid"))

# S4 Pooling Layer
model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="v
alid"))

# C5 Fully Connected Convolutional Layer
model.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation="t
anh", padding="valid"))
#Flatten the CNN output so that we can connect it with fully connected layers
model.add(layers.Flatten())

# FC6 Fully Connected Layer
model.add(layers.Dense(84, activation="tanh"))

#Output Layer with softmax activation
model.add(layers.Dense(133, activation="softmax"))

# Compile the model

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 224, 224, 6)       456
_________________________________________________________________
average_pooling2d_3 (Average (None, 223, 223, 6)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 219, 219, 16)      2416
_________________________________________________________________
average_pooling2d_4 (Average (None, 109, 109, 16)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 105, 105, 120)     48120
_________________________________________________________________
```

```
flatten_2 (Flatten)          (None, 1323000)           0
_________________________________________________________________
dense_3 (Dense)              (None, 84)                111132084
_________________________________________________________________
dense_4 (Dense)              (None, 133)               11305
=================================================================
Total params: 111,194,381
Trainable params: 111,194,381
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.callbacks import ModelCheckpoint
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=[
'accuracy'])
#model.compile(loss=keras.losses.categorical_crossentropy, optimizer="SGD", me
trics=["accuracy"])
#hist = model.fit(x=train_tensors,y=train_targets, epochs=10, batch_size=128,
 validation_data=(x_test, y_test), verbose=1)
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.from_scratc
h.hdf5',
                               verbose=1, save_best_only=True)

hist =model.fit(train_tensors, train_targets,
          validation_data=(valid_tensors,valid_targets),
          epochs=20, batch_size=256, callbacks=[checkpointer], verbose=1)
```

```
Train on 6680 samples, validate on 835 samples
Epoch 1/20
6656/6680 [============================>.] - ETA: 0s - loss: 5.0172 - acc: 0.0
065Epoch 00001: val_loss improved from inf to 4.90717, saving model to saved_m
odels/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 71s 11ms/step - loss: 5.0165 - ac
c: 0.0064 - val_loss: 4.9072 - val_acc: 0.0096
Epoch 2/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8890 - acc: 0.0
114Epoch 00002: val_loss improved from 4.90717 to 4.88528, saving model to sav
ed_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 59s 9ms/step - loss: 4.8888 - acc
: 0.0114 - val_loss: 4.8853 - val_acc: 0.0108
Epoch 3/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8852 - acc: 0.0
092Epoch 00003: val_loss improved from 4.88528 to 4.88348, saving model to sav
ed_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 59s 9ms/step - loss: 4.8851 - acc
: 0.0091 - val_loss: 4.8835 - val_acc: 0.0096
Epoch 4/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8851 - acc: 0.0
090Epoch 00004: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8849 - acc
: 0.0090 - val_loss: 4.8841 - val_acc: 0.0096
Epoch 5/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8845 - acc: 0.0
093Epoch 00005: val_loss improved from 4.88348 to 4.88197, saving model to sav
ed_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 59s 9ms/step - loss: 4.8849 - acc
: 0.0093 - val_loss: 4.8820 - val_acc: 0.0096
Epoch 6/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8826 - acc: 0.0
098Epoch 00006: val_loss did not improve
```

```
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8826 - acc
: 0.0097 - val_loss: 4.8851 - val_acc: 0.0096
Epoch 7/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8836 - acc: 0.0
096Epoch 00007: val_loss improved from 4.88197 to 4.87965, saving model to sav
ed_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 59s 9ms/step - loss: 4.8839 - acc
: 0.0096 - val_loss: 4.8797 - val_acc: 0.0096
Epoch 8/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8844 - acc: 0.0
107Epoch 00008: val_loss did not improve
6680/6680 [==============================] - 59s 9ms/step - loss: 4.8844 - acc
: 0.0108 - val_loss: 4.8822 - val_acc: 0.0108
Epoch 9/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8843 - acc: 0.0
096Epoch 00009: val_loss did not improve
6680/6680 [==============================] - 57s 9ms/step - loss: 4.8840 - acc
: 0.0096 - val_loss: 4.8855 - val_acc: 0.0108
Epoch 10/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8850 - acc: 0.0
087Epoch 00010: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8848 - acc
: 0.0087 - val_loss: 4.8826 - val_acc: 0.0108
Epoch 11/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8838 - acc: 0.0
081Epoch 00011: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8839 - acc
: 0.0081 - val_loss: 4.8838 - val_acc: 0.0108
Epoch 12/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8849 - acc: 0.0
086Epoch 00012: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8852 - acc
: 0.0085 - val_loss: 4.8799 - val_acc: 0.0096
Epoch 13/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8821 - acc: 0.0
095Epoch 00013: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8820 - acc
: 0.0094 - val_loss: 4.8859 - val_acc: 0.0108
Epoch 14/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8840 - acc: 0.0
092Epoch 00014: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8840 - acc
: 0.0093 - val_loss: 4.8843 - val_acc: 0.0096
Epoch 15/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8856 - acc: 0.0
090Epoch 00015: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8854 - acc
: 0.0090 - val_loss: 4.8829 - val_acc: 0.0096
Epoch 16/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8845 - acc: 0.0
090Epoch 00016: val_loss did not improve
6680/6680 [==============================] - 57s 9ms/step - loss: 4.8846 - acc
: 0.0090 - val_loss: 4.8816 - val_acc: 0.0096
Epoch 17/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8831 - acc: 0.0
072Epoch 00017: val_loss did not improve
6680/6680 [==============================] - 57s 9ms/step - loss: 4.8831 - acc
```

```
: 0.0072 - val_loss: 4.8856 - val_acc: 0.0108
Epoch 18/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8839 - acc: 0.0
081Epoch 00018: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8838 - acc
: 0.0081 - val_loss: 4.8808 - val_acc: 0.0096
Epoch 19/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8840 - acc: 0.0
095Epoch 00019: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8839 - acc
: 0.0096 - val_loss: 4.8855 - val_acc: 0.0108
Epoch 20/20
6656/6680 [============================>.] - ETA: 0s - loss: 4.8857 - acc: 0.0
105Epoch 00020: val_loss did not improve
6680/6680 [==============================] - 58s 9ms/step - loss: 4.8858 - acc
: 0.0105 - val_loss: 4.8814 - val_acc: 0.0096
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, 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('Test accuracy: %.4f%%' % test_accuracy)
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, 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('Test accuracy: %.4f%%' % test_accuracy)
Test accuracy: 1.0766%
```

In [ ]:

```
##GoogleNet 5
```

In [ ]:
```
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('/data/dog_images/train')
valid_files, valid_targets = load_dataset('/data/dog_images/valid')
test_files, test_targets = load_dataset('/data/dog_images/test')

# load list of dog names
dog_names = [item[20:-1] for item in sorted(glob("/data/dog_images/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))
#print( train_targets)
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.
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
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)
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 [01:27<00:00, 49.00it/s]
100%|██████████| 835/835 [00:09<00:00, 85.71it/s]
100%|██████████| 836/836 [00:09<00:00, 85.73it/s]
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D,MaxPool2
D,AveragePooling2D
from keras.layers import Dropout, Flatten, Dense, Input, concatenate
from keras.models import Sequential,Model

model = Sequential()
def inception_module(x,
                     filters_1x1,
                     filters_3x3_reduce,
                     filters_3x3,
                     filters_5x5_reduce,
                     filters_5x5,
                     filters_pool_proj,
                     name=None):

    conv_1x1 = Conv2D(filters_1x1, (1, 1), padding='same', activation='relu',
kernel_initializer=kernel_init, bias_initializer=bias_init)(x)

    conv_3x3 = Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation=
'relu', kernel_initializer=kernel_init, bias_initializer=bias_init)(x)
    conv_3x3 = Conv2D(filters_3x3, (3, 3), padding='same', activation='relu',
kernel_initializer=kernel_init, bias_initializer=bias_init)(conv_3x3)

    conv_5x5 = Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation=
'relu', kernel_initializer=kernel_init, bias_initializer=bias_init)(x)
    conv_5x5 = Conv2D(filters_5x5, (5, 5), padding='same', activation='relu',
kernel_initializer=kernel_init, bias_initializer=bias_init)(conv_5x5)

    pool_proj = MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
    pool_proj = Conv2D(filters_pool_proj, (1, 1), padding='same', activation=
'relu', kernel_initializer=kernel_init, bias_initializer=bias_init)(pool_proj)

    output = concatenate([conv_1x1, conv_3x3, conv_5x5, pool_proj], axis=3, na
me=name)

    return output
from keras import initializers
kernel_init = initializers.glorot_uniform()
bias_init = initializers.Constant(value=0.2)
input_layer = Input(shape=(224, 224, 3))

x = Conv2D(64, (7, 7), padding='same', strides=(2, 2), activation='relu', name
='conv_1_7x7/2', kernel_initializer=kernel_init, bias_initializer=bias_init)(i
```

```
nput_layer)
x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_1_3x3/2')
(x)
x = Conv2D(64, (1, 1), padding='same', strides=(1, 1), activation='relu', name
='conv_2a_3x3/1')(x)
x = Conv2D(192, (3, 3), padding='same', strides=(1, 1), activation='relu', nam
e='conv_2b_3x3/1')(x)
x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_2_3x3/2')
(x)

x = inception_module(x,
                     filters_1x1=64,
                     filters_3x3_reduce=96,
                     filters_3x3=128,
                     filters_5x5_reduce=16,
                     filters_5x5=32,
                     filters_pool_proj=32,
                     name='inception_3a')

x = inception_module(x,
                     filters_1x1=128,
                     filters_3x3_reduce=128,
                     filters_3x3=192,
                     filters_5x5_reduce=32,
                     filters_5x5=96,
                     filters_pool_proj=64,
                     name='inception_3b')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_3_3x3/2')
(x)

x = inception_module(x,
                     filters_1x1=192,
                     filters_3x3_reduce=96,
                     filters_3x3=208,
                     filters_5x5_reduce=16,
                     filters_5x5=48,
                     filters_pool_proj=64,
                     name='inception_4a')


x1 = AveragePooling2D((5, 5), strides=3)(x)
x1 = Conv2D(128, (1, 1), padding='same', activation='relu')(x1)
x1 = Flatten()(x1)
x1 = Dense(1024, activation='relu')(x1)
x1 = Dropout(0.7)(x1)
x1 = Dense(133, activation='softmax', name='auxilliary_output_1')(x1)

x = inception_module(x,
                     filters_1x1=160,
                     filters_3x3_reduce=112,
                     filters_3x3=224,
                     filters_5x5_reduce=24,
                     filters_5x5=64,
                     filters_pool_proj=64,
                     name='inception_4b')

```

```
x = inception_module(x,
                     filters_1x1=128,
                     filters_3x3_reduce=128,
                     filters_3x3=256,
                     filters_5x5_reduce=24,
                     filters_5x5=64,
                     filters_pool_proj=64,
                     name='inception_4c')

x = inception_module(x,
                     filters_1x1=112,
                     filters_3x3_reduce=144,
                     filters_3x3=288,
                     filters_5x5_reduce=32,
                     filters_5x5=64,
                     filters_pool_proj=64,
                     name='inception_4d')


x2 = AveragePooling2D((5, 5), strides=3)(x)
x2 = Conv2D(128, (1, 1), padding='same', activation='relu')(x2)
x2 = Flatten()(x2)
x2 = Dense(1024, activation='relu')(x2)
x2 = Dropout(0.7)(x2)
x2 = Dense(133, activation='softmax', name='auxilliary_output_2')(x2)

x = inception_module(x,
                     filters_1x1=256,
                     filters_3x3_reduce=160,
                     filters_3x3=320,
                     filters_5x5_reduce=32,
                     filters_5x5=128,
                     filters_pool_proj=128,
                     name='inception_4e')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max_pool_4_3x3/2')
(x)

x = inception_module(x,
                     filters_1x1=256,
                     filters_3x3_reduce=160,
                     filters_3x3=320,
                     filters_5x5_reduce=32,
                     filters_5x5=128,
                     filters_pool_proj=128,
                     name='inception_5a')

x = inception_module(x,
                     filters_1x1=384,
                     filters_3x3_reduce=192,
                     filters_3x3=384,
                     filters_5x5_reduce=48,
                     filters_5x5=128,
                     filters_pool_proj=128,
                     name='inception_5b')

x = GlobalAveragePooling2D(name='avg_pool_5_3x3/1')(x)
```

```
x = Dropout(0.4)(x)

x = Dense(133, activation='softmax', name='output')(x)
model = Model(input_layer, [x, x1, x2], name='inception_v1')
model.summary()
```

```
______________________________________________________________________________
____________________
Layer (type)                    Output Shape         Param #     Connected to
==============================================================================
====================
input_1 (InputLayer)            (None, 224, 224, 3)  0
______________________________________________________________________________
____________________
conv_1_7x7/2 (Conv2D)           (None, 112, 112, 64) 9472        input_1[0][0]
______________________________________________________________________________
____________________
max_pool_1_3x3/2 (MaxPooling2D) (None, 56, 56, 64)   0           conv_1_7x7/2[
0][0]
______________________________________________________________________________
____________________
conv_2a_3x3/1 (Conv2D)          (None, 56, 56, 64)   4160        max_pool_1_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv_2b_3x3/1 (Conv2D)          (None, 56, 56, 192)  110784      conv_2a_3x3/1
[0][0]
______________________________________________________________________________
____________________
max_pool_2_3x3/2 (MaxPooling2D) (None, 28, 28, 192)  0           conv_2b_3x3/1
[0][0]
______________________________________________________________________________
____________________
conv2d_2 (Conv2D)               (None, 28, 28, 96)   18528       max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_4 (Conv2D)               (None, 28, 28, 16)   3088        max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
____________________
max_pooling2d_1 (MaxPooling2D)  (None, 28, 28, 192)  0           max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_1 (Conv2D)               (None, 28, 28, 64)   12352       max_pool_2_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_3 (Conv2D)               (None, 28, 28, 128)  110720      conv2d_2[0][0
]
______________________________________________________________________________
____________________
conv2d_5 (Conv2D)               (None, 28, 28, 32)   12832       conv2d_4[0][0
]
______________________________________________________________________________
____________________
```

```
conv2d_6 (Conv2D)               (None, 28, 28, 32)   6176        max_pooling2d
_1[0][0]
__________________________________________________________________________________
________________________
inception_3a (Concatenate)      (None, 28, 28, 256)  0           conv2d_1[0][0
]
                                                                 conv2d_3[0][0
]
                                                                 conv2d_5[0][0
]
                                                                 conv2d_6[0][0
]
__________________________________________________________________________________
________________________
conv2d_8 (Conv2D)               (None, 28, 28, 128)  32896       inception_3a[
0][0]
__________________________________________________________________________________
________________________
conv2d_10 (Conv2D)              (None, 28, 28, 32)   8224        inception_3a[
0][0]
__________________________________________________________________________________
________________________
max_pooling2d_2 (MaxPooling2D)  (None, 28, 28, 256)  0           inception_3a[
0][0]
__________________________________________________________________________________
________________________
conv2d_7 (Conv2D)               (None, 28, 28, 128)  32896       inception_3a[
0][0]
__________________________________________________________________________________
________________________
conv2d_9 (Conv2D)               (None, 28, 28, 192)  221376      conv2d_8[0][0
]
__________________________________________________________________________________
________________________
conv2d_11 (Conv2D)              (None, 28, 28, 96)   76896       conv2d_10[0][
0]
__________________________________________________________________________________
________________________
conv2d_12 (Conv2D)              (None, 28, 28, 64)   16448       max_pooling2d
_2[0][0]
__________________________________________________________________________________
________________________
inception_3b (Concatenate)      (None, 28, 28, 480)  0           conv2d_7[0][0
]
                                                                 conv2d_9[0][0
]
                                                                 conv2d_11[0][
0]
                                                                 conv2d_12[0][
0]
__________________________________________________________________________________
________________________
max_pool_3_3x3/2 (MaxPooling2D) (None, 14, 14, 480)  0           inception_3b[
0][0]
__________________________________________________________________________________
________________________
conv2d_14 (Conv2D)              (None, 14, 14, 96)   46176       max_pool_3_3x
```

```
3/2[0][0]
________________________________________________________________________________
__________________
conv2d_16 (Conv2D)              (None, 14, 14, 16)   7696        max_pool_3_3x
3/2[0][0]
________________________________________________________________________________
__________________
max_pooling2d_3 (MaxPooling2D)  (None, 14, 14, 480)  0           max_pool_3_3x
3/2[0][0]
________________________________________________________________________________
__________________
conv2d_13 (Conv2D)              (None, 14, 14, 192)  92352       max_pool_3_3x
3/2[0][0]
________________________________________________________________________________
__________________
conv2d_15 (Conv2D)              (None, 14, 14, 208)  179920      conv2d_14[0][
0]
________________________________________________________________________________
__________________
conv2d_17 (Conv2D)              (None, 14, 14, 48)   19248       conv2d_16[0][
0]
________________________________________________________________________________
__________________
conv2d_18 (Conv2D)              (None, 14, 14, 64)   30784       max_pooling2d
_3[0][0]
________________________________________________________________________________
__________________
inception_4a (Concatenate)      (None, 14, 14, 512)  0           conv2d_13[0][
0]
                                                                 conv2d_15[0][
0]
                                                                 conv2d_17[0][
0]
                                                                 conv2d_18[0][
0]
________________________________________________________________________________
__________________
conv2d_21 (Conv2D)              (None, 14, 14, 112)  57456       inception_4a[
0][0]
________________________________________________________________________________
__________________
conv2d_23 (Conv2D)              (None, 14, 14, 24)   12312       inception_4a[
0][0]
________________________________________________________________________________
__________________
max_pooling2d_4 (MaxPooling2D)  (None, 14, 14, 512)  0           inception_4a[
0][0]
________________________________________________________________________________
__________________
conv2d_20 (Conv2D)              (None, 14, 14, 160)  82080       inception_4a[
0][0]
________________________________________________________________________________
__________________
conv2d_22 (Conv2D)              (None, 14, 14, 224)  226016      conv2d_21[0][
0]
________________________________________________________________________________
__________________
```

```
conv2d_24 (Conv2D)              (None, 14, 14, 64)   38464       conv2d_23[0][
0]
__________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 14, 14, 64)   32832       max_pooling2d
_4[0][0]
__________________________________________________________________________________________________
inception_4b (Concatenate)      (None, 14, 14, 512)  0           conv2d_20[0][
0]
                                                                 conv2d_22[0][
0]
                                                                 conv2d_24[0][
0]
                                                                 conv2d_25[0][
0]
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 14, 14, 128)  65664       inception_4b[
0][0]
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 14, 14, 24)   12312       inception_4b[
0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 14, 14, 512)  0           inception_4b[
0][0]
__________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 14, 14, 128)  65664       inception_4b[
0][0]
__________________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 14, 14, 256)  295168      conv2d_27[0][
0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 14, 14, 64)   38464       conv2d_29[0][
0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 14, 14, 64)   32832       max_pooling2d
_5[0][0]
__________________________________________________________________________________________________
inception_4c (Concatenate)      (None, 14, 14, 512)  0           conv2d_26[0][
0]
                                                                 conv2d_28[0][
0]
                                                                 conv2d_30[0][
0]
                                                                 conv2d_31[0][
0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 14, 14, 144)  73872       inception_4c[
```

```
0][0]
________________________________________________________________________________
__________________________
conv2d_35 (Conv2D)              (None, 14, 14, 32)   16416       inception_4c[
0][0]
________________________________________________________________________________
__________________________
max_pooling2d_6 (MaxPooling2D)  (None, 14, 14, 512)  0           inception_4c[
0][0]
________________________________________________________________________________
__________________________
conv2d_32 (Conv2D)              (None, 14, 14, 112)  57456       inception_4c[
0][0]
________________________________________________________________________________
__________________________
conv2d_34 (Conv2D)              (None, 14, 14, 288)  373536      conv2d_33[0][
0]
________________________________________________________________________________
__________________________
conv2d_36 (Conv2D)              (None, 14, 14, 64)   51264       conv2d_35[0][
0]
________________________________________________________________________________
__________________________
conv2d_37 (Conv2D)              (None, 14, 14, 64)   32832       max_pooling2d
_6[0][0]
________________________________________________________________________________
__________________________
inception_4d (Concatenate)      (None, 14, 14, 528)  0           conv2d_32[0][
0]
                                                                 conv2d_34[0][
0]
                                                                 conv2d_36[0][
0]
                                                                 conv2d_37[0][
0]
________________________________________________________________________________
__________________________
conv2d_40 (Conv2D)              (None, 14, 14, 160)  84640       inception_4d[
0][0]
________________________________________________________________________________
__________________________
conv2d_42 (Conv2D)              (None, 14, 14, 32)   16928       inception_4d[
0][0]
________________________________________________________________________________
__________________________
max_pooling2d_7 (MaxPooling2D)  (None, 14, 14, 528)  0           inception_4d[
0][0]
________________________________________________________________________________
__________________________
conv2d_39 (Conv2D)              (None, 14, 14, 256)  135424      inception_4d[
0][0]
________________________________________________________________________________
__________________________
conv2d_41 (Conv2D)              (None, 14, 14, 320)  461120      conv2d_40[0][
0]
________________________________________________________________________________
__________________________
```

```
conv2d_43 (Conv2D)              (None, 14, 14, 128)  102528      conv2d_42[0][
0]
______________________________________________________________________________
____________________
conv2d_44 (Conv2D)              (None, 14, 14, 128)  67712       max_pooling2d
_7[0][0]
______________________________________________________________________________
____________________
inception_4e (Concatenate)      (None, 14, 14, 832)  0           conv2d_39[0][
0]
                                                                 conv2d_41[0][
0]
                                                                 conv2d_43[0][
0]
                                                                 conv2d_44[0][
0]
______________________________________________________________________________
____________________
max_pool_4_3x3/2 (MaxPooling2D) (None, 7, 7, 832)    0           inception_4e[
0][0]
______________________________________________________________________________
____________________
conv2d_46 (Conv2D)              (None, 7, 7, 160)    133280      max_pool_4_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_48 (Conv2D)              (None, 7, 7, 32)     26656       max_pool_4_3x
3/2[0][0]
______________________________________________________________________________
____________________
max_pooling2d_8 (MaxPooling2D)  (None, 7, 7, 832)    0           max_pool_4_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_45 (Conv2D)              (None, 7, 7, 256)    213248      max_pool_4_3x
3/2[0][0]
______________________________________________________________________________
____________________
conv2d_47 (Conv2D)              (None, 7, 7, 320)    461120      conv2d_46[0][
0]
______________________________________________________________________________
____________________
conv2d_49 (Conv2D)              (None, 7, 7, 128)    102528      conv2d_48[0][
0]
______________________________________________________________________________
____________________
conv2d_50 (Conv2D)              (None, 7, 7, 128)    106624      max_pooling2d
_8[0][0]
______________________________________________________________________________
____________________
inception_5a (Concatenate)      (None, 7, 7, 832)    0           conv2d_45[0][
0]
                                                                 conv2d_47[0][
0]
                                                                 conv2d_49[0][
0]
                                                                 conv2d_50[0][
```

```
0]
________________________________________________________________________________
__________________
conv2d_52 (Conv2D)              (None, 7, 7, 192)    159936      inception_5a[
0][0]
________________________________________________________________________________
__________________
conv2d_54 (Conv2D)              (None, 7, 7, 48)     39984       inception_5a[
0][0]
________________________________________________________________________________
__________________
max_pooling2d_9 (MaxPooling2D)  (None, 7, 7, 832)    0           inception_5a[
0][0]
________________________________________________________________________________
__________________
average_pooling2d_1 (AveragePoo (None, 4, 4, 512)    0           inception_4a[
0][0]
________________________________________________________________________________
__________________
average_pooling2d_2 (AveragePoo (None, 4, 4, 528)    0           inception_4d[
0][0]
________________________________________________________________________________
__________________
conv2d_51 (Conv2D)              (None, 7, 7, 384)    319872      inception_5a[
0][0]
________________________________________________________________________________
__________________
conv2d_53 (Conv2D)              (None, 7, 7, 384)    663936      conv2d_52[0][
0]
________________________________________________________________________________
__________________
conv2d_55 (Conv2D)              (None, 7, 7, 128)    153728      conv2d_54[0][
0]
________________________________________________________________________________
__________________
conv2d_56 (Conv2D)              (None, 7, 7, 128)    106624      max_pooling2d
_9[0][0]
________________________________________________________________________________
__________________
conv2d_19 (Conv2D)              (None, 4, 4, 128)    65664       average_pooli
ng2d_1[0][0]
________________________________________________________________________________
__________________
conv2d_38 (Conv2D)              (None, 4, 4, 128)    67712       average_pooli
ng2d_2[0][0]
________________________________________________________________________________
__________________
inception_5b (Concatenate)      (None, 7, 7, 1024)   0           conv2d_51[0][
0]
                                                                 conv2d_53[0][
0]
                                                                 conv2d_55[0][
0]
                                                                 conv2d_56[0][
0]
________________________________________________________________________________
__________________
```

```
flatten_1 (Flatten)             (None, 2048)        0           conv2d_19[0][
0]
______________________________________________________________________________
____________________
flatten_2 (Flatten)             (None, 2048)        0           conv2d_38[0][
0]
______________________________________________________________________________
____________________
avg_pool_5_3x3/1 (GlobalAverage (None, 1024)        0           inception_5b[
0][0]
______________________________________________________________________________
____________________
dense_1 (Dense)                 (None, 1024)        2098176     flatten_1[0][
0]
______________________________________________________________________________
____________________
dense_2 (Dense)                 (None, 1024)        2098176     flatten_2[0][
0]
______________________________________________________________________________
____________________
dropout_3 (Dropout)             (None, 1024)        0           avg_pool_5_3x
3/1[0][0]
______________________________________________________________________________
____________________
dropout_1 (Dropout)             (None, 1024)        0           dense_1[0][0]
______________________________________________________________________________
____________________
dropout_2 (Dropout)             (None, 1024)        0           dense_2[0][0]
______________________________________________________________________________
____________________
output (Dense)                  (None, 133)         136325      dropout_3[0][
0]
______________________________________________________________________________
____________________
auxilliary_output_1 (Dense)     (None, 133)         136325      dropout_1[0][
0]
______________________________________________________________________________
____________________
auxilliary_output_2 (Dense)     (None, 133)         136325      dropout_2[0][
0]
==============================================================================
====================
Total params: 10,712,255
Trainable params: 10,712,255
Non-trainable params: 0
______________________________________________________________________________
____________________
```

```
from keras.optimizers import SGD
from keras.callbacks import LearningRateScheduler
epochs = 25
initial_lrate = 0.01

def decay(epoch, steps=100):
    initial_lrate = 0.01
    drop = 0.96
    epochs_drop = 8
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
```

```
    return lrate

sgd = SGD(lr=initial_lrate, momentum=0.9, nesterov=False)

lr_sc = LearningRateScheduler(decay)

#model.compile(loss=['categorical_crossentropy', 'categorical_crossentropy',
 'categorical_crossentropy'], loss_weights=[1, 0.3, 0.3], optimizer=sgd, metri
cs=['accuracy'])
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=[
'accuracy'])
from keras.callbacks import ModelCheckpoint

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

epochs = 20

### Do NOT modify the code below this line.

checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.from_scratc
h.hdf5',
                               verbose=1, save_best_only=True)

#model.fit(train_tensors, train_targets,
#          validation_data=(valid_tensors, valid_targets),
#          epochs=epochs, batch_size=20, callbacks=[checkpointer], verbose=1)
model.fit(train_tensors, [train_targets, train_targets,train_targets],
          validation_data=(valid_tensors,[valid_targets,valid_targets,valid_ta
rgets]),
          epochs=epochs, batch_size=20, callbacks=[checkpointer], verbose=1)
```

```
Train on 6680 samples, validate on 835 samples
Epoch 1/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6447 - output_
loss: 4.8881 - auxilliary_output_1_loss: 4.8776 - auxilliary_output_2_loss: 4.
8790 - output_acc: 0.0083 - auxilliary_output_1_acc: 0.0095 - auxilliary_outpu
t_2_acc: 0.0104Epoch 00001: val_loss improved from inf to 14.61411, saving mod
el to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6454 - o
utput_loss: 4.8883 - auxilliary_output_1_loss: 4.8779 - auxilliary_output_2_lo
ss: 4.8792 - output_acc: 0.0082 - auxilliary_output_1_acc: 0.0094 - auxilliary
_output_2_acc: 0.0105 - val_loss: 14.6141 - val_output_loss: 4.8759 - val_auxi
lliary_output_1_loss: 4.8690 - val_auxilliary_output_2_loss: 4.8692 - val_outp
ut_acc: 0.0096 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2
_acc: 0.0108
Epoch 2/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6310 - output_
loss: 4.8799 - auxilliary_output_1_loss: 4.8736 - auxilliary_output_2_loss: 4.
8775 - output_acc: 0.0099 - auxilliary_output_1_acc: 0.0098 - auxilliary_outpu
t_2_acc: 0.0113Epoch 00002: val_loss improved from 14.61411 to 14.61234, savin
g model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6310 - o
utput_loss: 4.8799 - auxilliary_output_1_loss: 4.8736 - auxilliary_output_2_lo
ss: 4.8775 - output_acc: 0.0099 - auxilliary_output_1_acc: 0.0097 - auxilliary
_output_2_acc: 0.0112 - val_loss: 14.6123 - val_output_loss: 4.8743 - val_auxi
lliary_output_1_loss: 4.8690 - val_auxilliary_output_2_loss: 4.8690 - val_outp
ut_acc: 0.0084 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2
```

```
_acc: 0.0108
Epoch 3/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6178 - output_loss: 4.8784 - auxilliary_output_1_loss: 4.8691 - auxilliary_output_2_loss: 4.8703 - output_acc: 0.0087 - auxilliary_output_1_acc: 0.0110 - auxilliary_output_2_acc: 0.0105Epoch 00003: val_loss improved from 14.61234 to 14.61001, saving model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6173 - output_loss: 4.8782 - auxilliary_output_1_loss: 4.8689 - auxilliary_output_2_loss: 4.8702 - output_acc: 0.0087 - auxilliary_output_1_acc: 0.0109 - auxilliary_output_2_acc: 0.0105 - val_loss: 14.6100 - val_output_loss: 4.8721 - val_auxilliary_output_1_loss: 4.8690 - val_auxilliary_output_2_loss: 4.8689 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 4/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6130 - output_loss: 4.8743 - auxilliary_output_1_loss: 4.8684 - auxilliary_output_2_loss: 4.8703 - output_acc: 0.0086 - auxilliary_output_1_acc: 0.0098 - auxilliary_output_2_acc: 0.0099Epoch 00004: val_loss improved from 14.61001 to 14.60823, saving model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6130 - output_loss: 4.8743 - auxilliary_output_1_loss: 4.8684 - auxilliary_output_2_loss: 4.8703 - output_acc: 0.0085 - auxilliary_output_1_acc: 0.0097 - auxilliary_output_2_acc: 0.0099 - val_loss: 14.6082 - val_output_loss: 4.8708 - val_auxilliary_output_1_loss: 4.8688 - val_auxilliary_output_2_loss: 4.8687 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 5/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6167 - output_loss: 4.8723 - auxilliary_output_1_loss: 4.8742 - auxilliary_output_2_loss: 4.8702 - output_acc: 0.0104 - auxilliary_output_1_acc: 0.0107 - auxilliary_output_2_acc: 0.0110Epoch 00005: val_loss improved from 14.60823 to 14.60783, saving model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6164 - output_loss: 4.8722 - auxilliary_output_1_loss: 4.8741 - auxilliary_output_2_loss: 4.8701 - output_acc: 0.0103 - auxilliary_output_1_acc: 0.0106 - auxilliary_output_2_acc: 0.0109 - val_loss: 14.6078 - val_output_loss: 4.8699 - val_auxilliary_output_1_loss: 4.8690 - val_auxilliary_output_2_loss: 4.8690 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 6/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6086 - output_loss: 4.8702 - auxilliary_output_1_loss: 4.8699 - auxilliary_output_2_loss: 4.8685 - output_acc: 0.0101 - auxilliary_output_1_acc: 0.0098 - auxilliary_output_2_acc: 0.0113Epoch 00006: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6086 - output_loss: 4.8702 - auxilliary_output_1_loss: 4.8699 - auxilliary_output_2_loss: 4.8685 - output_acc: 0.0100 - auxilliary_output_1_acc: 0.0097 - auxilliary_output_2_acc: 0.0112 - val_loss: 14.6108 - val_output_loss: 4.8732 - val_auxilliary_output_1_loss: 4.8689 - val_auxilliary_output_2_loss: 4.8687 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 7/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6125 - output_loss: 4.8689 - auxilliary_output_1_loss: 4.8685 - auxilliary_output_2_loss: 4.8751 - output_acc: 0.0108 - auxilliary_output_1_acc: 0.0110 - auxilliary_output_2_acc: 0.0098Epoch 00007: val_loss improved from 14.60783 to 14.60660, savin
```

```
g model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6128 - output_loss: 4.8690 - auxilliary_output_1_loss: 4.8686 - auxilliary_output_2_loss: 4.8752 - output_acc: 0.0108 - auxilliary_output_1_acc: 0.0109 - auxilliary_output_2_acc: 0.0097 - val_loss: 14.6066 - val_output_loss: 4.8691 - val_auxilliary_output_1_loss: 4.8687 - val_auxilliary_output_2_loss: 4.8688 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 8/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6127 - output_loss: 4.8696 - auxilliary_output_1_loss: 4.8722 - auxilliary_output_2_loss: 4.8709 - output_acc: 0.0110 - auxilliary_output_1_acc: 0.0107 - auxilliary_output_2_acc: 0.0114Epoch 00008: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6124 - output_loss: 4.8695 - auxilliary_output_1_loss: 4.8720 - auxilliary_output_2_loss: 4.8708 - output_acc: 0.0109 - auxilliary_output_1_acc: 0.0106 - auxilliary_output_2_acc: 0.0114 - val_loss: 14.6078 - val_output_loss: 4.8696 - val_auxilliary_output_1_loss: 4.8692 - val_auxilliary_output_2_loss: 4.8689 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 9/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6043 - output_loss: 4.8685 - auxilliary_output_1_loss: 4.8690 - auxilliary_output_2_loss: 4.8669 - output_acc: 0.0105 - auxilliary_output_1_acc: 0.0119 - auxilliary_output_2_acc: 0.0104Epoch 00009: val_loss improved from 14.60660 to 14.60644, saving model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6046 - output_loss: 4.8686 - auxilliary_output_1_loss: 4.8690 - auxilliary_output_2_loss: 4.8669 - output_acc: 0.0106 - auxilliary_output_1_acc: 0.0120 - auxilliary_output_2_acc: 0.0105 - val_loss: 14.6064 - val_output_loss: 4.8691 - val_auxilliary_output_1_loss: 4.8687 - val_auxilliary_output_2_loss: 4.8686 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 10/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6032 - output_loss: 4.8674 - auxilliary_output_1_loss: 4.8687 - auxilliary_output_2_loss: 4.8670 - output_acc: 0.0096 - auxilliary_output_1_acc: 0.0116 - auxilliary_output_2_acc: 0.0104Epoch 00010: val_loss improved from 14.60644 to 14.60636, saving model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6030 - output_loss: 4.8673 - auxilliary_output_1_loss: 4.8687 - auxilliary_output_2_loss: 4.8670 - output_acc: 0.0096 - auxilliary_output_1_acc: 0.0115 - auxilliary_output_2_acc: 0.0103 - val_loss: 14.6064 - val_output_loss: 4.8689 - val_auxilliary_output_1_loss: 4.8687 - val_auxilliary_output_2_loss: 4.8688 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 11/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.5995 - output_loss: 4.8671 - auxilliary_output_1_loss: 4.8652 - auxilliary_output_2_loss: 4.8672 - output_acc: 0.0111 - auxilliary_output_1_acc: 0.0113 - auxilliary_output_2_acc: 0.0119Epoch 00011: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.5989 - output_loss: 4.8669 - auxilliary_output_1_loss: 4.8651 - auxilliary_output_2_loss: 4.8670 - output_acc: 0.0111 - auxilliary_output_1_acc: 0.0112 - auxilliary_output_2_acc: 0.0118 - val_loss: 14.6065 - val_output_loss: 4.8686 - val_auxilliary_output_1_loss: 4.8689 - val_auxilliary_output_2_loss: 4.8690 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2
```

```
_acc: 0.0108
Epoch 12/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.5997 - output_
loss: 4.8660 - auxilliary_output_1_loss: 4.8670 - auxilliary_output_2_loss: 4.
8667 - output_acc: 0.0093 - auxilliary_output_1_acc: 0.0123 - auxilliary_outpu
t_2_acc: 0.0116Epoch 00012: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.5997 - o
utput_loss: 4.8660 - auxilliary_output_1_loss: 4.8669 - auxilliary_output_2_lo
ss: 4.8668 - output_acc: 0.0094 - auxilliary_output_1_acc: 0.0124 - auxilliary
_output_2_acc: 0.0117 - val_loss: 14.6068 - val_output_loss: 4.8693 - val_auxi
lliary_output_1_loss: 4.8687 - val_auxilliary_output_2_loss: 4.8688 - val_outp
ut_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2
_acc: 0.0108
Epoch 13/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.5984 - output_
loss: 4.8659 - auxilliary_output_1_loss: 4.8658 - auxilliary_output_2_loss: 4.
8666 - output_acc: 0.0101 - auxilliary_output_1_acc: 0.0119 - auxilliary_outpu
t_2_acc: 0.0111Epoch 00013: val_loss improved from 14.60636 to 14.60619, savin
g model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.5980 - o
utput_loss: 4.8659 - auxilliary_output_1_loss: 4.8657 - auxilliary_output_2_lo
ss: 4.8665 - output_acc: 0.0100 - auxilliary_output_1_acc: 0.0118 - auxilliary
_output_2_acc: 0.0111 - val_loss: 14.6062 - val_output_loss: 4.8689 - val_auxi
lliary_output_1_loss: 4.8686 - val_auxilliary_output_2_loss: 4.8687 - val_outp
ut_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2
_acc: 0.0108
Epoch 14/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6077 - output_
loss: 4.8672 - auxilliary_output_1_loss: 4.8671 - auxilliary_output_2_loss: 4.
8734 - output_acc: 0.0104 - auxilliary_output_1_acc: 0.0122 - auxilliary_outpu
t_2_acc: 0.0117Epoch 00014: val_loss improved from 14.60619 to 14.60593, savin
g model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6080 - o
utput_loss: 4.8673 - auxilliary_output_1_loss: 4.8672 - auxilliary_output_2_lo
ss: 4.8735 - output_acc: 0.0103 - auxilliary_output_1_acc: 0.0121 - auxilliary
_output_2_acc: 0.0117 - val_loss: 14.6059 - val_output_loss: 4.8688 - val_auxi
lliary_output_1_loss: 4.8686 - val_auxilliary_output_2_loss: 4.8686 - val_outp
ut_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2
_acc: 0.0108
Epoch 15/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.5965 - output_
loss: 4.8651 - auxilliary_output_1_loss: 4.8650 - auxilliary_output_2_loss: 4.
8665 - output_acc: 0.0114 - auxilliary_output_1_acc: 0.0114 - auxilliary_outpu
t_2_acc: 0.0113Epoch 00015: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.5971 - o
utput_loss: 4.8653 - auxilliary_output_1_loss: 4.8651 - auxilliary_output_2_lo
ss: 4.8667 - output_acc: 0.0114 - auxilliary_output_1_acc: 0.0114 - auxilliary
_output_2_acc: 0.0112 - val_loss: 14.6087 - val_output_loss: 4.8711 - val_auxi
lliary_output_1_loss: 4.8690 - val_auxilliary_output_2_loss: 4.8686 - val_outp
ut_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2
_acc: 0.0108
Epoch 16/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6061 - output_
loss: 4.8666 - auxilliary_output_1_loss: 4.8669 - auxilliary_output_2_loss: 4.
8727 - output_acc: 0.0120 - auxilliary_output_1_acc: 0.0129 - auxilliary_outpu
t_2_acc: 0.0117Epoch 00016: val_loss improved from 14.60593 to 14.60559, savin
g model to saved_models/weights.best.from_scratch.hdf5
```

```
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6063 - output_loss: 4.8667 - auxilliary_output_1_loss: 4.8669 - auxilliary_output_2_loss: 4.8727 - output_acc: 0.0120 - auxilliary_output_1_acc: 0.0129 - auxilliary_output_2_acc: 0.0117 - val_loss: 14.6056 - val_output_loss: 4.8685 - val_auxilliary_output_1_loss: 4.8686 - val_auxilliary_output_2_loss: 4.8685 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 17/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.5957 - output_loss: 4.8652 - auxilliary_output_1_loss: 4.8650 - auxilliary_output_2_loss: 4.8656 - output_acc: 0.0117 - auxilliary_output_1_acc: 0.0113 - auxilliary_output_2_acc: 0.0114Epoch 00017: val_loss improved from 14.60559 to 14.60557, saving model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 91s 14ms/step - loss: 14.5963 - output_loss: 4.8654 - auxilliary_output_1_loss: 4.8652 - auxilliary_output_2_loss: 4.8658 - output_acc: 0.0117 - auxilliary_output_1_acc: 0.0112 - auxilliary_output_2_acc: 0.0114 - val_loss: 14.6056 - val_output_loss: 4.8685 - val_auxilliary_output_1_loss: 4.8685 - val_auxilliary_output_2_loss: 4.8686 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 18/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.5964 - output_loss: 4.8652 - auxilliary_output_1_loss: 4.8649 - auxilliary_output_2_loss: 4.8663 - output_acc: 0.0111 - auxilliary_output_1_acc: 0.0114 - auxilliary_output_2_acc: 0.0114Epoch 00018: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.5962 - output_loss: 4.8652 - auxilliary_output_1_loss: 4.8648 - auxilliary_output_2_loss: 4.8662 - output_acc: 0.0112 - auxilliary_output_1_acc: 0.0115 - auxilliary_output_2_acc: 0.0115 - val_loss: 14.6212 - val_output_loss: 4.8840 - val_auxilliary_output_1_loss: 4.8686 - val_auxilliary_output_2_loss: 4.8686 - val_output_acc: 0.0096 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 19/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.5986 - output_loss: 4.8657 - auxilliary_output_1_loss: 4.8653 - auxilliary_output_2_loss: 4.8676 - output_acc: 0.0108 - auxilliary_output_1_acc: 0.0111 - auxilliary_output_2_acc: 0.0113Epoch 00019: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.5979 - output_loss: 4.8654 - auxilliary_output_1_loss: 4.8651 - auxilliary_output_2_loss: 4.8673 - output_acc: 0.0111 - auxilliary_output_1_acc: 0.0114 - auxilliary_output_2_acc: 0.0115 - val_loss: 14.6060 - val_output_loss: 4.8688 - val_auxilliary_output_1_loss: 4.8686 - val_auxilliary_output_2_loss: 4.8686 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
Epoch 20/20
6660/6680 [============================>.] - ETA: 0s - loss: 14.6003 - output_loss: 4.8651 - auxilliary_output_1_loss: 4.8703 - auxilliary_output_2_loss: 4.8649 - output_acc: 0.0111 - auxilliary_output_1_acc: 0.0114 - auxilliary_output_2_acc: 0.0117Epoch 00020: val_loss did not improve
6680/6680 [==============================] - 91s 14ms/step - loss: 14.6003 - output_loss: 4.8652 - auxilliary_output_1_loss: 4.8703 - auxilliary_output_2_loss: 4.8649 - output_acc: 0.0111 - auxilliary_output_1_acc: 0.0114 - auxilliary_output_2_acc: 0.0117 - val_loss: 14.6061 - val_output_loss: 4.8687 - val_auxilliary_output_1_loss: 4.8689 - val_auxilliary_output_2_loss: 4.8686 - val_output_acc: 0.0108 - val_auxilliary_output_1_acc: 0.0108 - val_auxilliary_output_2_acc: 0.0108
<keras.callbacks.History at 0x7f8b926e2e48>
```

```
model.load_weights('saved_models/weights.best.from_scratch.hdf5')
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, 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('Test accuracy: %.4f%%' % test_accuracy)
Test accuracy: 0.0000%
```

In [ ]:

```
I have tried GoogleNet 5 ,LeNet and LeNet5 and LeNet 5 has highest accuray but
compare to sugested model ot has much much parameter
than sugested one So for performace i chose the sugested one for simplest and
according to my cpu it is faster
```

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 [18]:

```
bottleneck_features = np.load('/data/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 [19]:
```
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_3 ( (None, 512)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 133)               68229     
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
_________________________________________________________________
```

Compile the Model

In [20]:
```
VGG16_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metr
ics=['accuracy'])
```

Train the Model

In [21]:
```
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: 0s - loss: 12.9514 - acc:
0.1044Epoch 00001: val_loss improved from inf to 11.54031, saving model to sa
ved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 338us/step - loss: 12.9433 -
acc: 0.1048 - val_loss: 11.5403 - val_acc: 0.1952
Epoch 2/20
6660/6680 [============================>.] - ETA: 0s - loss: 11.2073 - acc:
0.2336Epoch 00002: val_loss improved from 11.54031 to 11.01780, saving model
to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 266us/step - loss: 11.2054 -
acc: 0.2337 - val_loss: 11.0178 - val_acc: 0.2431
Epoch 3/20
6460/6680 [============================>.] - ETA: 0s - loss: 10.7185 - acc:
0.2834Epoch 00003: val_loss improved from 11.01780 to 10.82729, saving model
to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 266us/step - loss: 10.7285 -
acc: 0.2832 - val_loss: 10.8273 - val_acc: 0.2563
Epoch 4/20
6560/6680 [============================>.] - ETA: 0s - loss: 10.4754 - acc:
0.3145Epoch 00004: val_loss improved from 10.82729 to 10.67333, saving model
to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 264us/step - loss: 10.4868 -
acc: 0.3141 - val_loss: 10.6733 - val_acc: 0.2802
Epoch 5/20
6460/6680 [============================>.] - ETA: 0s - loss: 10.4038 - acc:
0.3297Epoch 00005: val_loss did not improve
6680/6680 [==============================] - 2s 269us/step - loss: 10.4020 -
acc: 0.3292 - val_loss: 10.6848 - val_acc: 0.2922
Epoch 6/20
6520/6680 [============================>.] - ETA: 0s - loss: 10.2926 - acc:
0.3396Epoch 00006: val_loss improved from 10.67333 to 10.50736, saving model
to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 266us/step - loss: 10.2933 -
acc: 0.3388 - val_loss: 10.5074 - val_acc: 0.2922
Epoch 7/20
6480/6680 [============================>.] - ETA: 0s - loss: 10.0086 - acc:
0.3535Epoch 00007: val_loss improved from 10.50736 to 10.20887, saving model
to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 267us/step - loss: 10.0283 -
acc: 0.3521 - val_loss: 10.2089 - val_acc: 0.3234
Epoch 8/20
6480/6680 [============================>.] - ETA: 0s - loss: 9.9248 - acc: 0.
3633Epoch 00008: val_loss improved from 10.20887 to 10.15315, saving model to
saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 267us/step - loss: 9.8819 - a
cc: 0.3650 - val_loss: 10.1532 - val_acc: 0.3246
Epoch 9/20
6560/6680 [============================>.] - ETA: 0s - loss: 9.7705 - acc: 0.
3758Epoch 00009: val_loss improved from 10.15315 to 10.14385, saving model to
saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 262us/step - loss: 9.7608 - a
cc: 0.3762 - val_loss: 10.1439 - val_acc: 0.3222
Epoch 10/20
6560/6680 [============================>.] - ETA: 0s - loss: 9.7160 - acc: 0.
3841Epoch 00010: val_loss did not improve
```

```
6680/6680 [==============================] - 2s 261us/step - loss: 9.7154 - a
cc: 0.3844 - val_loss: 10.1591 - val_acc: 0.3198
Epoch 11/20
6460/6680 [============================>.] - ETA: 0s - loss: 9.6825 - acc: 0.
3896Epoch 00011: val_loss improved from 10.14385 to 10.13183, saving model to
saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 267us/step - loss: 9.6996 - a
cc: 0.3877 - val_loss: 10.1318 - val_acc: 0.3293
Epoch 12/20
6460/6680 [============================>.] - ETA: 0s - loss: 9.6429 - acc: 0.
3944Epoch 00012: val_loss improved from 10.13183 to 10.10367, saving model to
saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 267us/step - loss: 9.6784 - a
cc: 0.3922 - val_loss: 10.1037 - val_acc: 0.3317
Epoch 13/20
6660/6680 [============================>.] - ETA: 0s - loss: 9.6637 - acc: 0.
3941Epoch 00013: val_loss did not improve
6680/6680 [==============================] - 2s 262us/step - loss: 9.6638 - a
cc: 0.3942 - val_loss: 10.1175 - val_acc: 0.3329
Epoch 14/20
6520/6680 [============================>.] - ETA: 0s - loss: 9.5708 - acc: 0.
3957Epoch 00014: val_loss improved from 10.10367 to 10.06252, saving model to
saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 265us/step - loss: 9.5516 - a
cc: 0.3972 - val_loss: 10.0625 - val_acc: 0.3305
Epoch 15/20
6580/6680 [============================>.] - ETA: 0s - loss: 9.4896 - acc: 0.
4035Epoch 00015: val_loss improved from 10.06252 to 9.93964, saving model to
saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 262us/step - loss: 9.4878 - a
cc: 0.4036 - val_loss: 9.9396 - val_acc: 0.3377
Epoch 16/20
6560/6680 [============================>.] - ETA: 0s - loss: 9.4139 - acc: 0.
4084Epoch 00016: val_loss improved from 9.93964 to 9.89933, saving model to s
aved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 265us/step - loss: 9.3935 - a
cc: 0.4093 - val_loss: 9.8993 - val_acc: 0.3413
Epoch 17/20
6480/6680 [============================>.] - ETA: 0s - loss: 9.3309 - acc: 0.
4119Epoch 00017: val_loss improved from 9.89933 to 9.79955, saving model to s
aved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 263us/step - loss: 9.3152 - a
cc: 0.4129 - val_loss: 9.7996 - val_acc: 0.3389
Epoch 18/20
6540/6680 [============================>.] - ETA: 0s - loss: 9.2134 - acc: 0.
4188Epoch 00018: val_loss improved from 9.79955 to 9.72994, saving model to s
aved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 264us/step - loss: 9.2110 - a
cc: 0.4190 - val_loss: 9.7299 - val_acc: 0.3437
Epoch 19/20
6480/6680 [============================>.] - ETA: 0s - loss: 9.1427 - acc: 0.
4228Epoch 00019: val_loss did not improve
6680/6680 [==============================] - 2s 264us/step - loss: 9.1347 - a
cc: 0.4228 - val_loss: 9.7915 - val_acc: 0.3389
Epoch 20/20
6560/6680 [============================>.] - ETA: 0s - loss: 9.0479 - acc: 0.
4265Epoch 00020: val_loss improved from 9.72994 to 9.64054, saving model to s
```

```
aved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 2s 262us/step - loss: 9.0449 - a
cc: 0.4268 - val_loss: 9.6405 - val_acc: 0.3461
```

Out[21]:

```
<keras.callbacks.History at 0x7f03345687f0>
```

Load the Model with the Best Validation Loss

In [22]:

```
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 [23]:

```
# 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(VGG16_predictions)
print('Test accuracy: %.4f%%' % test_accuracy)
```

```
Test accuracy: 34.5694%
```

Predict Dog Breed with the Model

In [24]:

```
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. These are already in the workspace, at /data/bottleneck_features. If you wish to download them on a different machine, they can be found at:

- 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`.

The above architectures are downloaded and stored for you in the `/data/bottleneck_features/` folder.

This means the following will be in the `/data/bottleneck_features/` folder:

`DogVGG19Data.npz` `DogResnet50Data.npz` `DogInceptionV3Data.npz` `DogXceptionData.npz`

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

In [25]:
```
### TODO: Obtain bottleneck features from another pre-trained CNN.
network = 'Xception'
if network =='ResNet-50':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogResnet
50Data.npz')
elif network == 'Inception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogIncept
ionV3Data.npz')
elif network =='Xception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogXcepti
onData.npz')
elif network =='VGG-19':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogVGG19D
ata.npz')

train_network = bottleneck_features_network['train']
valid_network = bottleneck_features_network['valid']
test_network = bottleneck_features_network['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:

In [27]:
```
### TODO: Define your architecture.
Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=train_network.shape[1
:]))
Xception_model.add(Dense(133, activation='softmax'))
```

(IMPLEMENTATION) Compile the Model

In [28]:
```
### TODO: Compile the model.
Xception_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

(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)](https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html), but this is not a requirement.

In [31]:
```
### TODO: Train the model.
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hd
f5',
                               verbose=1, save_best_only=True)

Xception_model.fit(train_network, train_targets,
          validation_data=(valid_network, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)
```

```
Train on 6680 samples, validate on 835 samples
Epoch 1/20
6660/6680 [============================>.] - ETA: 0s - loss: 1.0589Epoch 0000
1: val_loss improved from inf to 0.52986, saving model to saved_models/weight
s.best.Xception.hdf5
6680/6680 [==============================] - 4s 529us/step - loss: 1.0572 - v
al_loss: 0.5299
Epoch 2/20
6620/6680 [============================>.] - ETA: 0s - loss: 0.3999Epoch 0000
2: val_loss improved from 0.52986 to 0.47970, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 3s 522us/step - loss: 0.3989 - v
al_loss: 0.4797
Epoch 3/20
6560/6680 [============================>.] - ETA: 0s - loss: 0.3206Epoch 0000
3: val_loss improved from 0.47970 to 0.47003, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 4s 551us/step - loss: 0.3205 - v
al_loss: 0.4700
Epoch 4/20
6540/6680 [============================>.] - ETA: 0s - loss: 0.2768Epoch 0000
4: val_loss did not improve
6680/6680 [==============================] - 3s 459us/step - loss: 0.2766 - v
al_loss: 0.5070
Epoch 5/20
6560/6680 [============================>.] - ETA: 0s - loss: 0.2419Epoch 0000
5: val_loss did not improve
6680/6680 [==============================] - 3s 495us/step - loss: 0.2448 - v
al_loss: 0.4961
Epoch 6/20
6620/6680 [============================>.] - ETA: 0s - loss: 0.2193Epoch 0000
6: val_loss did not improve
6680/6680 [==============================] - 3s 480us/step - loss: 0.2186 - v
al_loss: 0.5260
Epoch 7/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.1948Epoch 0000
7: val_loss did not improve
6680/6680 [==============================] - 3s 505us/step - loss: 0.1962 - v
al_loss: 0.5316
Epoch 8/20
6620/6680 [============================>.] - ETA: 0s - loss: 0.1812Epoch 0000
8: val_loss did not improve
6680/6680 [==============================] - 3s 516us/step - loss: 0.1806 - v
al_loss: 0.5732
Epoch 9/20
6540/6680 [============================>.] - ETA: 0s - loss: 0.1597Epoch 0000
9: val_loss did not improve
6680/6680 [==============================] - 3s 482us/step - loss: 0.1598 - v
al_loss: 0.5945
Epoch 10/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.1497Epoch 0001
0: val_loss did not improve
6680/6680 [==============================] - 3s 454us/step - loss: 0.1496 - v
al_loss: 0.6041
Epoch 11/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.1371Epoch 0001
1: val_loss did not improve
```

```
6680/6680 [==============================] - 3s 457us/step - loss: 0.1376 - v
al_loss: 0.5959
Epoch 12/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.1248Epoch 0001
2: val_loss did not improve
6680/6680 [==============================] - 3s 503us/step - loss: 0.1246 - v
al_loss: 0.6231
Epoch 13/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.1176Epoch 0001
3: val_loss did not improve
6680/6680 [==============================] - 3s 417us/step - loss: 0.1177 - v
al_loss: 0.6275
Epoch 14/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.1098Epoch 0001
4: val_loss did not improve
6680/6680 [==============================] - 3s 428us/step - loss: 0.1100 - v
al_loss: 0.5983
Epoch 15/20
6620/6680 [============================>.] - ETA: 0s - loss: 0.0982Epoch 0001
5: val_loss did not improve
6680/6680 [==============================] - 3s 480us/step - loss: 0.0982 - v
al_loss: 0.6509
Epoch 16/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0944Epoch 0001
6: val_loss did not improve
6680/6680 [==============================] - 3s 468us/step - loss: 0.0940 - v
al_loss: 0.6499
Epoch 17/20
6540/6680 [============================>.] - ETA: 0s - loss: 0.0840Epoch 0001
7: val_loss did not improve
6680/6680 [==============================] - 3s 442us/step - loss: 0.0848 - v
al_loss: 0.6474
Epoch 18/20
6580/6680 [============================>.] - ETA: 0s - loss: 0.0836Epoch 0001
8: val_loss did not improve
6680/6680 [==============================] - 3s 431us/step - loss: 0.0827 - v
al_loss: 0.6645
Epoch 19/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0766Epoch 0001
9: val_loss did not improve
6680/6680 [==============================] - 3s 455us/step - loss: 0.0762 - v
al_loss: 0.6906
Epoch 20/20
6580/6680 [============================>.] - ETA: 0s - loss: 0.0728Epoch 0002
0: val_loss did not improve
6680/6680 [==============================] - 3s 423us/step - loss: 0.0734 - v
al_loss: 0.6937
```

Out[31]:
```
<keras.callbacks.History at 0x7f033425a5c0>
```

(IMPLEMENTATION) Load the Model with the Best Validation Loss

In [32]:
```python
### TODO: Load the model weights with the best validation loss.
Xception_model.load_weights('saved_models/weights.best.Xception.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 [34]:
```
### TODO: Calculate classification accuracy on the test dataset.

Xception_predictions = [np.argmax(Xception_model.predict(np.expand_dims(featur
e, axis=0))) for feature in test_network]

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

```
Test accuracy: 84.4498%
```

In [ ]:
```
### ResNet-50
```

In [35]:
```
### TODO: Obtain bottleneck features from another pre-trained CNN.
network = 'ResNet-50'
if network =='ResNet-50':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogResnet
50Data.npz')
elif network == 'Inception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogIncept
ionV3Data.npz')
elif network =='Xception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogXcepti
onData.npz')
elif network =='VGG-19':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogVGG19D
ata.npz')

train_network = bottleneck_features_network['train']
valid_network = bottleneck_features_network['valid']
test_network = bottleneck_features_network['test']

### TODO: Define your architecture.
Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=train_network.shape[1
:]))
Xception_model.add(Dense(133, activation='softmax'))

### TODO: Compile the model.
Xception_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

### TODO: Train the model.
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hd
f5',
                               verbose=1, save_best_only=True)

Xception_model.fit(train_network, train_targets,
          validation_data=(valid_network, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)

### TODO: Load the model weights with the best validation loss.
Xception_model.load_weights('saved_models/weights.best.Xception.hdf5')

### TODO: Calculate classification accuracy on the test dataset.

Xception_predictions = [np.argmax(Xception_model.predict(np.expand_dims(featur
e, axis=0))) for feature in test_network]

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

```
Train on 6680 samples, validate on 835 samples
Epoch 1/20
6520/6680 [============================>.] - ETA: 0s - loss: 1.6661Epoch 0000
1: val_loss improved from inf to 0.78518, saving model to saved_models/weight
s.best.Xception.hdf5
6680/6680 [==============================] - 2s 279us/step - loss: 1.6478 - v
al_loss: 0.7852
Epoch 2/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.4326Epoch 0000
2: val_loss improved from 0.78518 to 0.74410, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 231us/step - loss: 0.4327 - v
al_loss: 0.7441
Epoch 3/20
6520/6680 [============================>.] - ETA: 0s - loss: 0.2659Epoch 0000
3: val_loss improved from 0.74410 to 0.66815, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 225us/step - loss: 0.2644 - v
al_loss: 0.6682
Epoch 4/20
6520/6680 [============================>.] - ETA: 0s - loss: 0.1812Epoch 0000
4: val_loss improved from 0.66815 to 0.65669, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 1s 224us/step - loss: 0.1800 - v
al_loss: 0.6567
Epoch 5/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.1176Epoch 0000
5: val_loss did not improve
6680/6680 [==============================] - 1s 222us/step - loss: 0.1180 - v
al_loss: 0.6636
Epoch 6/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0867Epoch 0000
6: val_loss did not improve
6680/6680 [==============================] - 1s 221us/step - loss: 0.0866 - v
al_loss: 0.6889
Epoch 7/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.0605Epoch 0000
7: val_loss did not improve
6680/6680 [==============================] - 1s 220us/step - loss: 0.0606 - v
al_loss: 0.7447
Epoch 8/20
6580/6680 [============================>.] - ETA: 0s - loss: 0.0481Epoch 0000
8: val_loss did not improve
6680/6680 [==============================] - 1s 221us/step - loss: 0.0477 - v
al_loss: 0.7012
Epoch 9/20
6500/6680 [============================>.] - ETA: 0s - loss: 0.0325Epoch 0000
9: val_loss did not improve
6680/6680 [==============================] - 1s 223us/step - loss: 0.0323 - v
al_loss: 0.7491
Epoch 10/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.0275Epoch 0001
0: val_loss did not improve
6680/6680 [==============================] - 1s 220us/step - loss: 0.0274 - v
al_loss: 0.7608
Epoch 11/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.0205Epoch 0001
```

```
1: val_loss did not improve
6680/6680 [==============================] - 1s 221us/step - loss: 0.0205 - v
al_loss: 0.7138
Epoch 12/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.0156Epoch 0001
2: val_loss did not improve
6680/6680 [==============================] - 1s 221us/step - loss: 0.0155 - v
al_loss: 0.7803
Epoch 13/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.0131Epoch 0001
3: val_loss did not improve
6680/6680 [==============================] - 1s 222us/step - loss: 0.0131 - v
al_loss: 0.8263
Epoch 14/20
6580/6680 [============================>.] - ETA: 0s - loss: 0.0118Epoch 0001
4: val_loss did not improve
6680/6680 [==============================] - 1s 223us/step - loss: 0.0116 - v
al_loss: 0.8319
Epoch 15/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.0098Epoch 0001
5: val_loss did not improve
6680/6680 [==============================] - 1s 221us/step - loss: 0.0102 - v
al_loss: 0.8675
Epoch 16/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0085Epoch 0001
6: val_loss did not improve
6680/6680 [==============================] - 2s 226us/step - loss: 0.0084 - v
al_loss: 0.8593
Epoch 17/20
6560/6680 [============================>.] - ETA: 0s - loss: 0.0065Epoch 0001
7: val_loss did not improve
6680/6680 [==============================] - 1s 223us/step - loss: 0.0064 - v
al_loss: 0.8560
Epoch 18/20
6440/6680 [===========================>..] - ETA: 0s - loss: 0.0057Epoch 0001
8: val_loss did not improve
6680/6680 [==============================] - 1s 218us/step - loss: 0.0067 - v
al_loss: 0.9047
Epoch 19/20
6520/6680 [============================>.] - ETA: 0s - loss: 0.0069Epoch 0001
9: val_loss did not improve
6680/6680 [==============================] - 1s 224us/step - loss: 0.0068 - v
al_loss: 0.9122
Epoch 20/20
6460/6680 [============================>.] - ETA: 0s - loss: 0.0062Epoch 0002
0: val_loss did not improve
6680/6680 [==============================] - 1s 224us/step - loss: 0.0060 - v
al_loss: 0.9160
Test accuracy: 81.2201%
```

In [ ]:

```
#VGG-19
```

In [36]:
```
### TODO: Obtain bottleneck features from another pre-trained CNN.
network = 'VGG-19'
if network =='ResNet-50':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogResnet
50Data.npz')
elif network == 'Inception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogIncept
ionV3Data.npz')
elif network =='Xception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogXcepti
onData.npz')
elif network =='VGG-19':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogVGG19D
ata.npz')

train_network = bottleneck_features_network['train']
valid_network = bottleneck_features_network['valid']
test_network = bottleneck_features_network['test']

### TODO: Define your architecture.
Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=train_network.shape[1
:]))
Xception_model.add(Dense(133, activation='softmax'))

### TODO: Compile the model.
Xception_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

### TODO: Train the model.
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hd
f5',
                               verbose=1, save_best_only=True)

Xception_model.fit(train_network, train_targets,
          validation_data=(valid_network, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)

### TODO: Load the model weights with the best validation loss.
Xception_model.load_weights('saved_models/weights.best.Xception.hdf5')

### TODO: Calculate classification accuracy on the test dataset.

Xception_predictions = [np.argmax(Xception_model.predict(np.expand_dims(featur
e, axis=0))) for feature in test_network]

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

```
Train on 6680 samples, validate on 835 samples
Epoch 1/20
6540/6680 [============================>.] - ETA: 0s - loss: 12.2815Epoch 000
01: val_loss improved from inf to 10.73971, saving model to saved_models/weig
hts.best.Xception.hdf5
6680/6680 [==============================] - 2s 321us/step - loss: 12.2570 -
val_loss: 10.7397
Epoch 2/20
6580/6680 [============================>.] - ETA: 0s - loss: 10.1915Epoch 000
02: val_loss improved from 10.73971 to 10.20255, saving model to saved_model
s/weights.best.Xception.hdf5
6680/6680 [==============================] - 2s 258us/step - loss: 10.1919 -
val_loss: 10.2026
Epoch 3/20
6480/6680 [============================>.] - ETA: 0s - loss: 9.7654Epoch 0000
3: val_loss improved from 10.20255 to 9.98180, saving model to saved_models/w
eights.best.Xception.hdf5
6680/6680 [==============================] - 2s 254us/step - loss: 9.7887 - v
al_loss: 9.9818
Epoch 4/20
6620/6680 [============================>.] - ETA: 0s - loss: 9.5521Epoch 0000
4: val_loss improved from 9.98180 to 9.78641, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 257us/step - loss: 9.5453 - v
al_loss: 9.7864
Epoch 5/20
6620/6680 [============================>.] - ETA: 0s - loss: 9.2765Epoch 0000
5: val_loss improved from 9.78641 to 9.63579, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 259us/step - loss: 9.2875 - v
al_loss: 9.6358
Epoch 6/20
6600/6680 [============================>.] - ETA: 0s - loss: 9.0250Epoch 0000
6: val_loss improved from 9.63579 to 9.29517, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 258us/step - loss: 9.0101 - v
al_loss: 9.2952
Epoch 7/20
6520/6680 [============================>.] - ETA: 0s - loss: 8.7664Epoch 0000
7: val_loss improved from 9.29517 to 9.25001, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 253us/step - loss: 8.7722 - v
al_loss: 9.2500
Epoch 8/20
6600/6680 [============================>.] - ETA: 0s - loss: 8.6278Epoch 0000
8: val_loss improved from 9.25001 to 9.11837, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 256us/step - loss: 8.6314 - v
al_loss: 9.1184
Epoch 9/20
6440/6680 [===========================>..] - ETA: 0s - loss: 8.5083Epoch 0000
9: val_loss improved from 9.11837 to 9.03088, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 256us/step - loss: 8.5287 - v
al_loss: 9.0309
Epoch 10/20
6620/6680 [============================>.] - ETA: 0s - loss: 8.3500Epoch 0001
```

```
0: val_loss improved from 9.03088 to 8.86494, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 260us/step - loss: 8.3479 - v
al_loss: 8.8649
Epoch 11/20
6480/6680 [============================>.] - ETA: 0s - loss: 8.1487Epoch 0001
1: val_loss improved from 8.86494 to 8.77736, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 255us/step - loss: 8.1509 - v
al_loss: 8.7774
Epoch 12/20
6580/6680 [============================>.] - ETA: 0s - loss: 8.0856Epoch 0001
2: val_loss improved from 8.77736 to 8.71500, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 262us/step - loss: 8.0876 - v
al_loss: 8.7150
Epoch 13/20
6660/6680 [============================>.] - ETA: 0s - loss: 7.9739Epoch 0001
3: val_loss improved from 8.71500 to 8.48897, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 256us/step - loss: 7.9719 - v
al_loss: 8.4890
Epoch 14/20
6580/6680 [============================>.] - ETA: 0s - loss: 7.6841Epoch 0001
4: val_loss improved from 8.48897 to 8.33156, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 253us/step - loss: 7.6711 - v
al_loss: 8.3316
Epoch 15/20
6620/6680 [============================>.] - ETA: 0s - loss: 7.5583Epoch 0001
5: val_loss improved from 8.33156 to 8.27690, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 259us/step - loss: 7.5566 - v
al_loss: 8.2769
Epoch 16/20
6660/6680 [============================>.] - ETA: 0s - loss: 7.5042Epoch 0001
6: val_loss improved from 8.27690 to 8.20567, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 255us/step - loss: 7.5035 - v
al_loss: 8.2057
Epoch 17/20
6660/6680 [============================>.] - ETA: 0s - loss: 7.4657Epoch 0001
7: val_loss improved from 8.20567 to 8.12725, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 257us/step - loss: 7.4650 - v
al_loss: 8.1272
Epoch 18/20
6460/6680 [============================>.] - ETA: 0s - loss: 7.3376Epoch 0001
8: val_loss improved from 8.12725 to 8.01355, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 256us/step - loss: 7.3710 - v
al_loss: 8.0135
Epoch 19/20
6600/6680 [============================>.] - ETA: 0s - loss: 7.2494Epoch 0001
9: val_loss improved from 8.01355 to 7.96707, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 260us/step - loss: 7.2626 - v
```

```
al_loss: 7.9671
Epoch 20/20
6620/6680 [============================>.] - ETA: 0s - loss: 7.1952Epoch 0002
0: val_loss improved from 7.96707 to 7.87576, saving model to saved_models/we
ights.best.Xception.hdf5
6680/6680 [==============================] - 2s 257us/step - loss: 7.1982 - v
al_loss: 7.8758
Test accuracy: 45.2153%
```

In [37]:
```
### TODO: Obtain bottleneck features from another pre-trained CNN.
network = 'Inception'
if network =='ResNet-50':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogResnet
50Data.npz')
elif network == 'Inception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogIncept
ionV3Data.npz')
elif network =='Xception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogXcepti
onData.npz')
elif network =='VGG-19':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogVGG19D
ata.npz')

train_network = bottleneck_features_network['train']
valid_network = bottleneck_features_network['valid']
test_network = bottleneck_features_network['test']

### TODO: Define your architecture.
Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=train_network.shape[1
:]))
Xception_model.add(Dense(133, activation='softmax'))

### TODO: Compile the model.
Xception_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

### TODO: Train the model.
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hd
f5',
                               verbose=1, save_best_only=True)

Xception_model.fit(train_network, train_targets,
          validation_data=(valid_network, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)

### TODO: Load the model weights with the best validation loss.
Xception_model.load_weights('saved_models/weights.best.Xception.hdf5')

### TODO: Calculate classification accuracy on the test dataset.

Xception_predictions = [np.argmax(Xception_model.predict(np.expand_dims(featur
e, axis=0))) for feature in test_network]

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

```
Train on 6680 samples, validate on 835 samples
Epoch 1/20
6660/6680 [============================>.] - ETA: 0s - loss: 1.1626Epoch 0000
1: val_loss improved from inf to 0.60340, saving model to saved_models/weight
s.best.Xception.hdf5
6680/6680 [==============================] - 3s 465us/step - loss: 1.1608 - v
al_loss: 0.6034
Epoch 2/20
6620/6680 [============================>.] - ETA: 0s - loss: 0.4734Epoch 0000
2: val_loss did not improve
6680/6680 [==============================] - 2s 367us/step - loss: 0.4732 - v
al_loss: 0.6488
Epoch 3/20
6500/6680 [============================>.] - ETA: 0s - loss: 0.3637Epoch 0000
3: val_loss did not improve
6680/6680 [==============================] - 2s 318us/step - loss: 0.3652 - v
al_loss: 0.7619
Epoch 4/20
6540/6680 [============================>.] - ETA: 0s - loss: 0.2872Epoch 0000
4: val_loss did not improve
6680/6680 [==============================] - 2s 315us/step - loss: 0.2876 - v
al_loss: 0.7154
Epoch 5/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.2352Epoch 0000
5: val_loss did not improve
6680/6680 [==============================] - 2s 319us/step - loss: 0.2349 - v
al_loss: 0.6942
Epoch 6/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.2028Epoch 0000
6: val_loss did not improve
6680/6680 [==============================] - 2s 322us/step - loss: 0.2029 - v
al_loss: 0.7500
Epoch 7/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.1754Epoch 0000
7: val_loss did not improve
6680/6680 [==============================] - 2s 320us/step - loss: 0.1755 - v
al_loss: 0.7009
Epoch 8/20
6560/6680 [============================>.] - ETA: 0s - loss: 0.1393Epoch 0000
8: val_loss did not improve
6680/6680 [==============================] - 2s 370us/step - loss: 0.1409 - v
al_loss: 0.7681
Epoch 9/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.1246Epoch 0000
9: val_loss did not improve
6680/6680 [==============================] - 2s 369us/step - loss: 0.1238 - v
al_loss: 0.7771
Epoch 10/20
6540/6680 [============================>.] - ETA: 0s - loss: 0.1115Epoch 0001
0: val_loss did not improve
6680/6680 [==============================] - 2s 371us/step - loss: 0.1104 - v
al_loss: 0.7821
Epoch 11/20
6520/6680 [============================>.] - ETA: 0s - loss: 0.0902Epoch 0001
1: val_loss did not improve
6680/6680 [==============================] - 2s 367us/step - loss: 0.0917 - v
al_loss: 0.8221
```

```
Epoch 12/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0841Epoch 0001
2: val_loss did not improve
6680/6680 [==============================] - 2s 350us/step - loss: 0.0842 - v
al_loss: 0.8468
Epoch 13/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0689Epoch 0001
3: val_loss did not improve
6680/6680 [==============================] - 2s 335us/step - loss: 0.0687 - v
al_loss: 0.8661
Epoch 14/20
6600/6680 [============================>.] - ETA: 0s - loss: 0.0674Epoch 0001
4: val_loss did not improve
6680/6680 [==============================] - 2s 332us/step - loss: 0.0668 - v
al_loss: 0.8636
Epoch 15/20
6540/6680 [============================>.] - ETA: 0s - loss: 0.0546Epoch 0001
5: val_loss did not improve
6680/6680 [==============================] - 2s 336us/step - loss: 0.0555 - v
al_loss: 0.9089
Epoch 16/20
6560/6680 [============================>.] - ETA: 0s - loss: 0.0511Epoch 0001
6: val_loss did not improve
6680/6680 [==============================] - 2s 336us/step - loss: 0.0508 - v
al_loss: 0.8676
Epoch 17/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0392Epoch 0001
7: val_loss did not improve
6680/6680 [==============================] - 2s 341us/step - loss: 0.0393 - v
al_loss: 0.9199
Epoch 18/20
6660/6680 [============================>.] - ETA: 0s - loss: 0.0404Epoch 0001
8: val_loss did not improve
6680/6680 [==============================] - 2s 370us/step - loss: 0.0402 - v
al_loss: 0.9048
Epoch 19/20
6640/6680 [============================>.] - ETA: 0s - loss: 0.0382Epoch 0001
9: val_loss did not improve
6680/6680 [==============================] - 2s 340us/step - loss: 0.0380 - v
al_loss: 0.9022
Epoch 20/20
6520/6680 [============================>.] - ETA: 0s - loss: 0.0349Epoch 0002
0: val_loss did not improve
6680/6680 [==============================] - 2s 328us/step - loss: 0.0348 - v
al_loss: 0.9164
Test accuracy: 80.0239%
```

In [39]:
```
##best One xception
Xception : Test accuracy: 84.4498%
ResNet-50 : Test accuracy: 81.2201%
VGG-19 :  Test accuracy: 45.2153%
Inception :Test accuracy: 80.0239%
the Xception is selecetd beacouse has hieghst accuracy
```

In [ ]:

In [21]:
```
### TODO: Obtain bottleneck features from another pre-trained CNN.
network = 'Xception'
if network =='ResNet-50':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogResnet
50Data.npz')
elif network == 'Inception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogIncept
ionV3Data.npz')
elif network =='Xception':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogXcepti
onData.npz')
elif network =='VGG-19':
    bottleneck_features_network = np.load('/data/bottleneck_features/DogVGG19D
ata.npz')

train_network = bottleneck_features_network['train']
valid_network = bottleneck_features_network['valid']
test_network = bottleneck_features_network['test']

### TODO: Define your architecture.
Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=train_network.shape[1
:]))
Xception_model.add(Dense(133, activation='softmax'))

### TODO: Compile the model.
Xception_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
### TODO: Train the model.
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hd
f5',
                               verbose=1, save_best_only=True)

Xception_model.fit(train_network, train_targets,
          validation_data=(valid_network, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)

### TODO: Load the model weights with the best validation loss.
Xception_model.load_weights('saved_models/weights.best.Xception.hdf5')

### TODO: Calculate classification accuracy on the test dataset.

Xception_predictions = [np.argmax(Xception_model.predict(np.expand_dims(featur
e, axis=0))) for feature in test_network]

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

(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 [ ]:

```
```

In [20]:

```
### TODO: Write a function that takes a path to an image as input
### and returns the dog breed that is predicted by the model.
from extract_bottleneck_features import *
def Xception_predict_breed(img_path):
    # extract bottleneck features
    bottleneck_feature = extract_Xception(path_to_tensor(img_path))
    # obtain predicted vector
    predicted_vector = Xception_model.predict(bottleneck_feature)
    # return dog breed that is predicted by the model
    return dog_names[np.argmax(predicted_vector)]
```

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!

Sample Human Output

(IMPLEMENTATION) Write your Algorithm

In [5]:

```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.
def detect_breed(image_path):
    if face_detector(image_path):
        print("Hello, human!")
    elif dog_detector(image_path):
        print("Hello, dog!")
    else:
        print("Error: Neither a human face or a dog was detected.\n")
        return
    # Use same Image Pipeline as used earlier
    img = cv2.imread(image_path)
    # Convert from BGR to RGB
    cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Plot the
    plt.imshow(cv_rgb)
    plt.show()

    print("You look like a ...")
    print(Xception_predict_breed(image_path))
    print()
```

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.

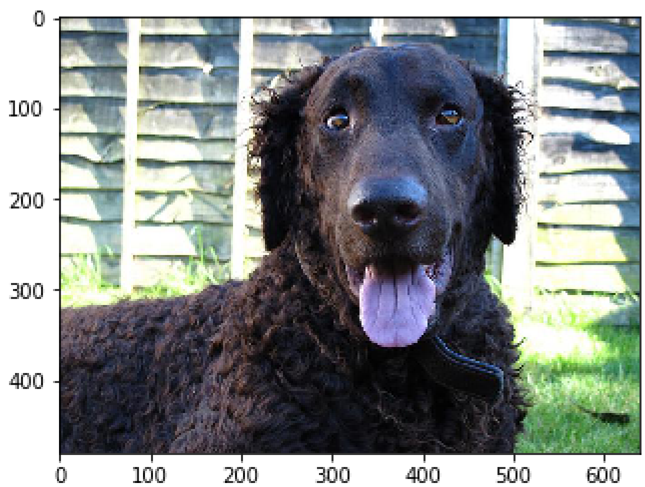
Answer:

In [26]:
```
## 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.
import cv2
import matplotlib.pyplot as plt
detect_breed("images/American_water_spaniel_00648.jpg")
#detect_breed("images/Brittany_02625.jpg.jpg")
detect_breed("images/Curly-coated_retriever_03896.jpg")
detect_breed("images/Labrador_retriever_06449.jpg")
detect_breed("images/Labrador_retriever_06455.jpg")
detect_breed("images/Labrador_retriever_06457.jpg")
#detect_breed("images/sample_human_output.png")
detect_breed("images/Welsh_springer_spaniel_08203.jpg")
```

Hello, dog!

You look like a ...
in/045.Cardigan_welsh_corgi

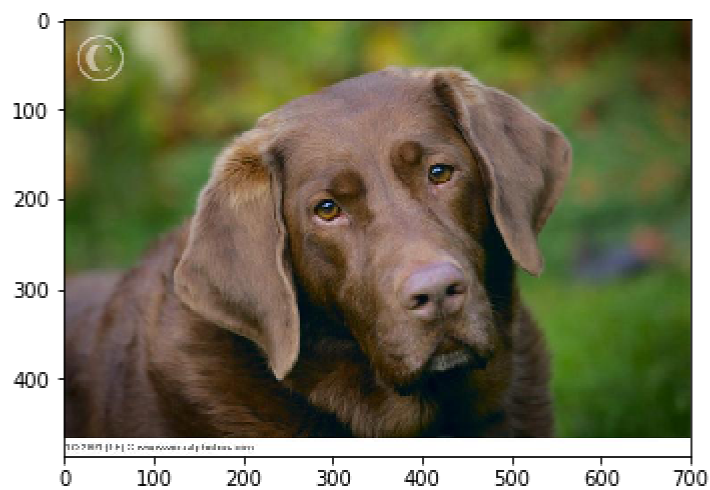
Hello, dog!



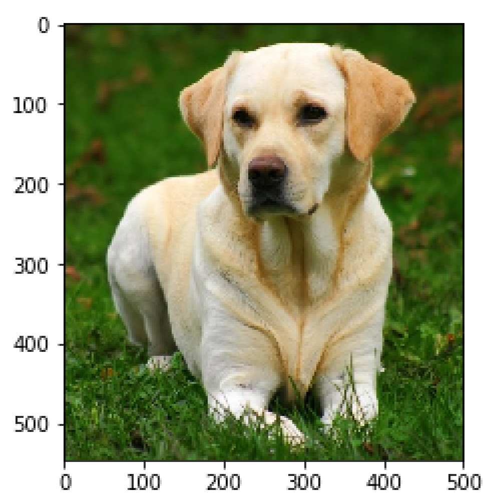
You look like a ...
in/083.Ibizan_hound

Hello, dog!

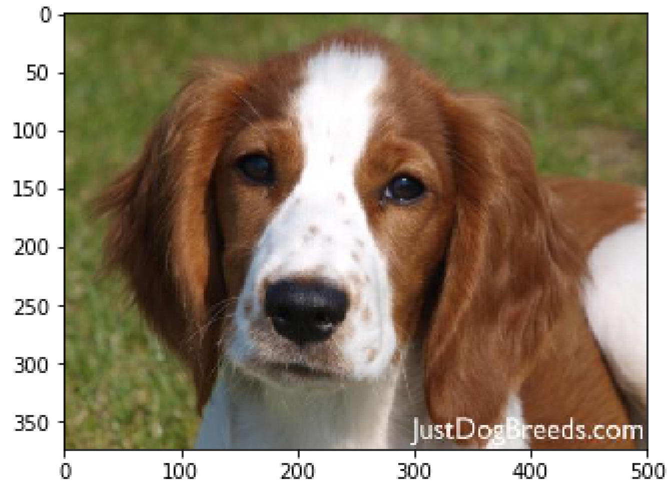

```
You look like a ...
in/060.Dogue_de_bordeaux

Hello, dog!
```




```
You look like a ...
in/060.Dogue_de_bordeaux

Hello, dog!
```



```
You look like a ...
in/060.Dogue_de_bordeaux

Hello, dog!
```




```
You look like a ...
in/068.Flat-coated_retriever
```

it seems my alghorithm is not succesful on defining the breed of dogs I need help and advise to improve my alghorithm ¶

Please download your notebook to submit

In order to submit, please do the following:

1. Download an HTML version of the notebook to your computer using 'File: Download as...'
2. Click on the orange Jupyter circle on the top left of the workspace.
3. Navigate into the dog-project folder to ensure that you are using the provided dog_images, lfw, and bottleneck_features folders; this means that those folders will *not* appear in the dog-project folder. If they do appear because you downloaded them, delete them.
4. While in the dog-project folder, upload the HTML version of this notebook you just downloaded. The upload button is on the top right.
5. Navigate back to the home folder by clicking on the two dots next to the folder icon, and then open up a terminal under the 'new' tab on the top right
6. Zip the dog-project folder with the following command in the terminal: `zip -r dog-project.zip dog-project`
7. Download the zip file by clicking on the square next to it and selecting 'download'. This will be the zip file you turn in on the next node after this workspace!