UDACITY CAPSTONE PROJECT PROPOSAL

ILYA PANSHENSKOV

Dog Breed Classification with CNN in Keras

Domain Background

The proposed problem lies in the domain of computer vision, the key challenge of which is to make computers derive information from visual input similar to the way the human brain processes visual information. Usually, this is achieved by highly sophisticated algorithms, which can quickly and accurately detect and label objects displayed. This field of computer science has recently received a lot of attention due to its wide commercial application (autonomous driving, healthcare, agriculture, etc).

Computer vision as a concept was coined in the 1960-s together with an emerging field of artificial intelligence. Most of the initial groundwork lay the foundation for image processing – making the image encoded in a numeric form. Around the same time the concept of a "pixel" – the idea that each digital image can be broken down into a sequence of numbers encoding position and color, came to be. The work of detecting an image was made mostly by a set of rule-based algorithms.

The next big leap for computer vision happened in the 1980-s when computer scientists started applying statistical approaches to the problem. They used some of the statistical models (linear regression, logistic regression, decision trees, or support vector machines) to classify images and objects within the image.

Three key innovations advanced computer vision in recent years:

– vast amount of visual data – images and videos available online, but most importantly labeled images. One of the most prominent datasets that was instrumental in the field of computer vision is ImageNet maintained by Stanford University and containing over 14 million classified images,

– deep learning techniques, proposed and promoted by researchers such as Geoffrey Hinton and Yann LeCun and adjusted for the computer vision problem in 2010-s,

– high-performance computing systems, such as GPUs or large-scale distributed clusters, which were able to process data on a big scale

Problem Statement

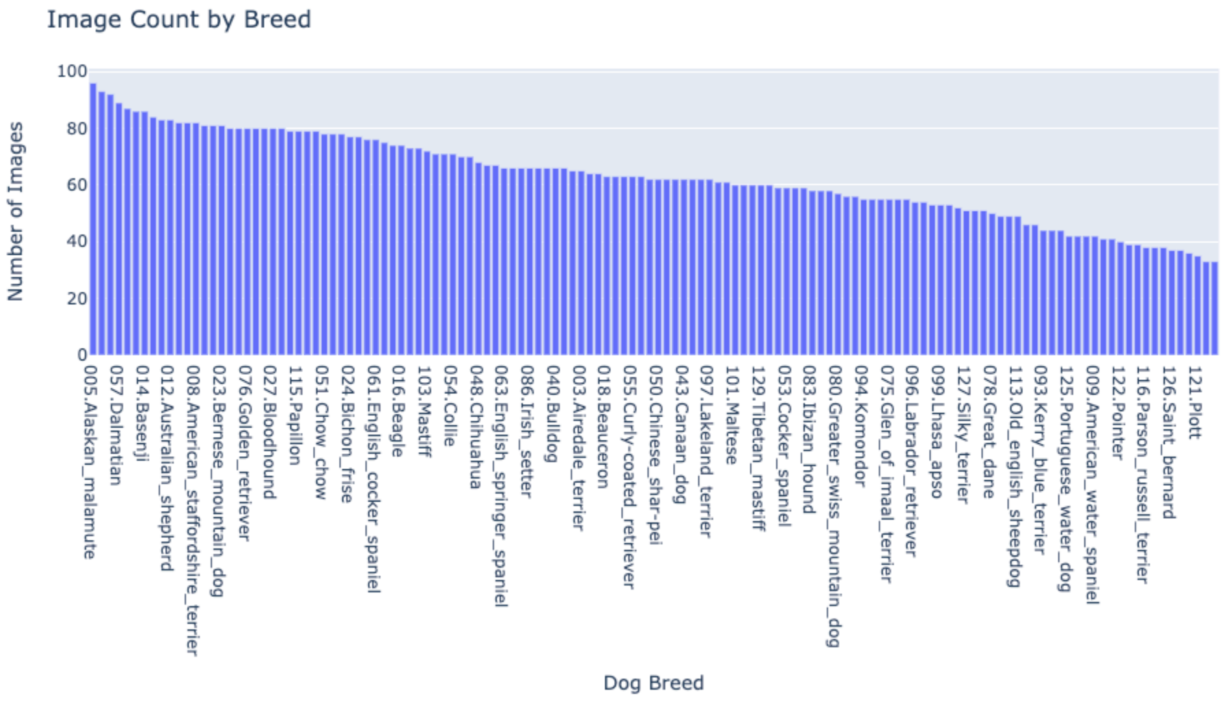
Dog breed classification from image data is the problem that we face in the proposed project. It is the one that has been mostly solved but to advance personal skillset we will tackle it from scratch. Given an image we would like the algorithm to detect:

1. Whether there is a dog in the picture.
2. If there is a dog – what breed that dog is.

Datasets and Inputs

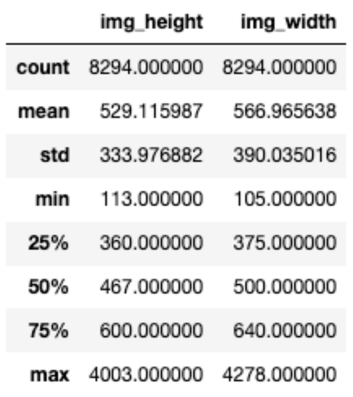
For the task on hand, we will be using the dataset embedded in Udacity workspace on Jupyter. The original dataset was compiled by Stanford's Computer Vision lab and consists of 20,580 color images of dogs (available for public access on the lab's website). The dataset in the Udacity Workspace includes 8,351 color images of dogs belonging to 133 different breeds. When parsing the data we observed a slight anomaly with the files belonging to Leonberger breed and decided to remove them. Thus the final number of images reduced to 8,294.

On this note it is worth mentioning that the dataset is slightly imbalanced, which means that there are more classes of certain breeds than the other as can be seen on the chart below:

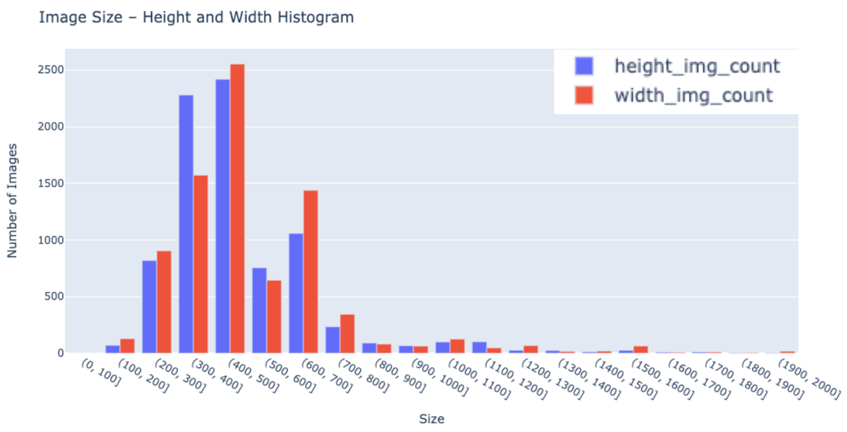


For example, there are almost three times as many images of Alaskan Malamute as of Xoloitzcuintli. This skewness of data makes us very cautious about what metric should be chosen when training and evaluating the model.

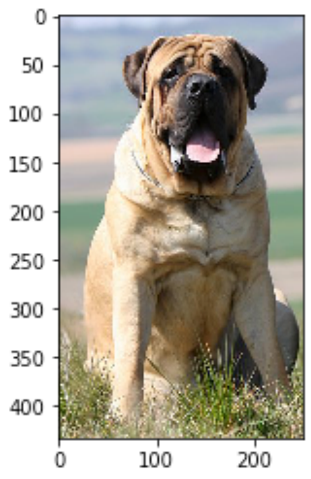
Analyzing the sizes of images revealed a large variance of widths and heights as well, the distribution of both being skewed to the right with a median being around 470 for height and 500 for width. Therefore when processing images we will normalize each picture to the size of 500 x 500 since the CNN model expects all data points to have the same shape.

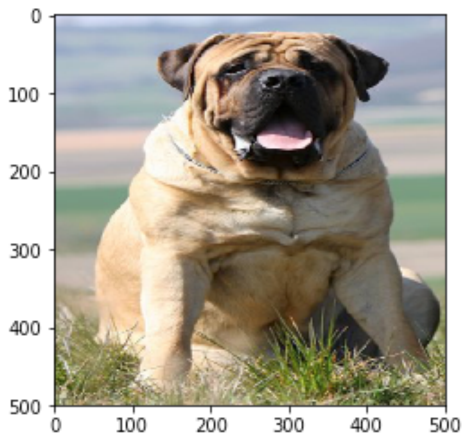


	img_height	img_width
count	8294.000000	8294.000000
mean	529.115987	566.965638
std	333.976882	390.035016
min	113.000000	105.000000
25%	360.000000	375.000000
50%	467.000000	500.000000
75%	600.000000	640.000000
max	4003.000000	4278.000000



This implies that this original image of a Mastiff presented on the left would actually be resized to the way it is displayed on the right.





The dataset in the Workspace consists of train, validation, and test subsets. As we will describe in more detail below, the training will be done with Keras high-level Python API, which has an embedded split into train and validation. For this reason, the input data will consist both of unlabeled train and validation data points. For prediction and evaluation purposes we will use the images in the test subset.

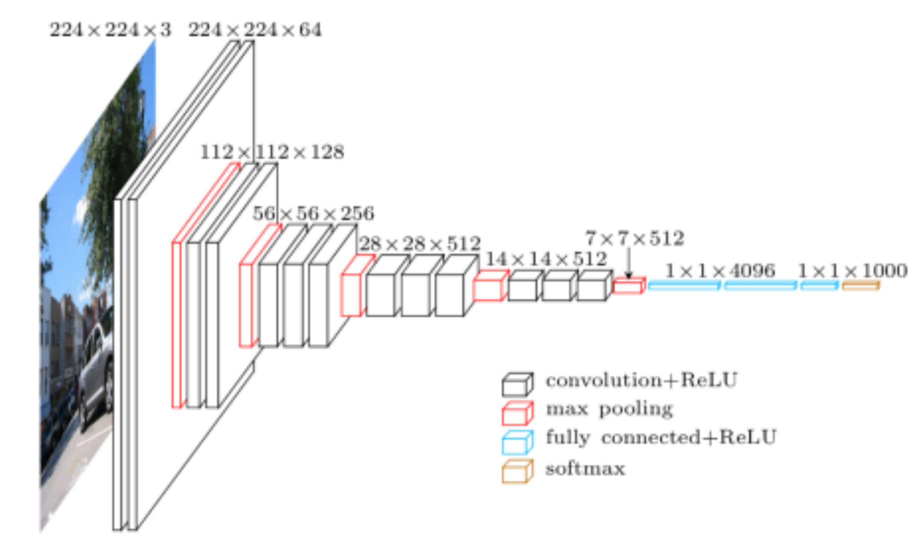
Solution Statement and Project Design

The solution to the problem has two parts: image processing and image classification.

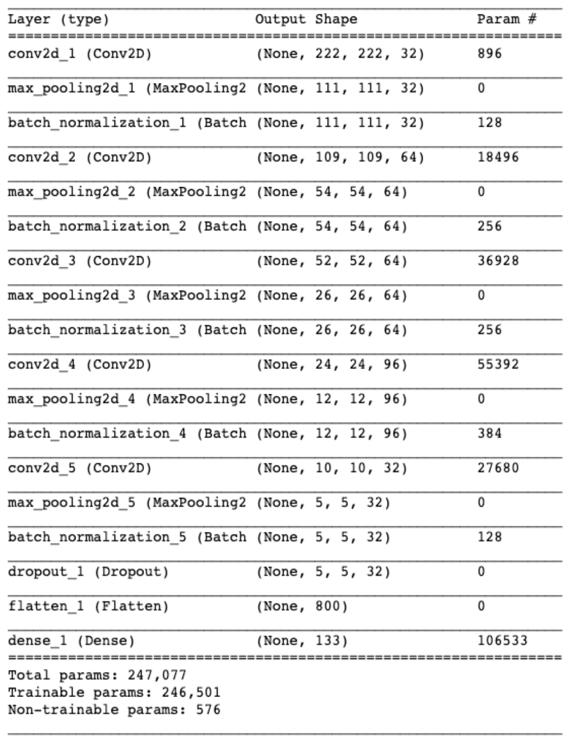
Image processing will be done mostly with Keras and Scipy's ndimage library. Numpy and Pandas Python libraries will be used to read images into numerical data and store it. Each image will be presented as a three-dimensional numpy array since the images have color. As discussed above, the images should have the same pixel size, which will be set to 500 by 500.

One of the issues with the image processing and classification, which is not limited to the given dataset is statistical noise, i.e. the number of objects in the picture that does not belong to the class. Consider the image on the left. Although the Chihuahua is clearly positioned in the center, the image includes another dog and two people. This might be a problem for the model, the clear solution to which is unclear. We might label the objects in the image by analyzing the intensity of pixels, then – splitting the image into several images.

For the purpose of image classification we will be using Convolutional Neural Network done exclusively in Keras. The architecture of the model is inspired by the VGG16 model's architecture, outlined in 2015 Simonyan and Zisserman paper. The VGG16 received its name after the concept of a "very deep Convolutional Network" that consists of 16 layers. The most advantage of the VGG16 model is a set of "very small 3x3 convolution" filters that can elicit image features, followed by maxpool layers of 2x2 filter of stride 2. In the end it has two dense fully connected layers followed by a softmax functon for output.



Although this architecture has proven to be very efficient for the job of image classification, a big disadvantage of it is the large number of of parameters (approximately 138 million). That means that the training of the model would require both enormous amount of computational power and time.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 32)      896       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 32)      0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 111, 111, 32)      128       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 109, 109, 64)      18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 64)        0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 54, 54, 64)        256       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 52, 52, 64)        36928     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 64)        0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 26, 26, 64)        256       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 96)        55392     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 96)        0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 12, 12, 96)        384       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 32)        27680     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 32)          0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 32)          128       
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 32)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 133)               106533    
=================================================================
Total params: 247,077
Trainable params: 246,501
Non-trainable params: 576
_________________________________________________________________
```

We would like to propose a "light-weighted" version of VGG16, which would use a set of Conv2D layers. To mitigate the issue of excessive number of parameters (and by extension – overfitting), we will add max pool layers after each convolution and also drop out layer. Each convolutional layer would be activated through ReLU function. The output of the neural network will be designed as a Dense layer with Softmax function.

The ultimate challenge is to design the model that would perform well both on train and validation subsets, the metrics for which (log loss and accuracy) we will plot to derive the number of epochs sufficient for training.

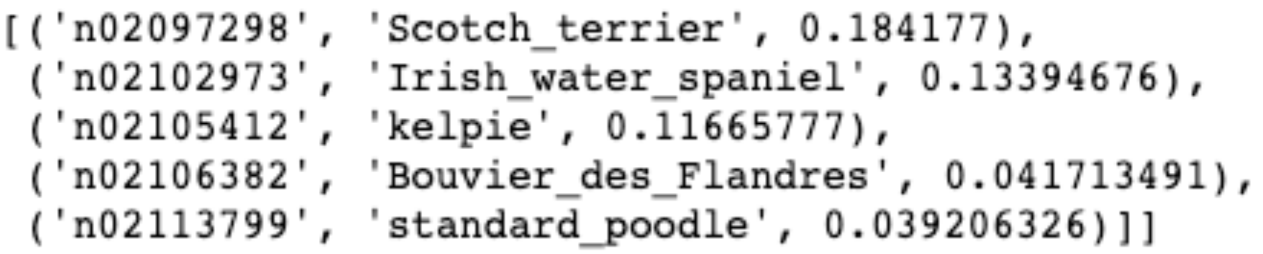
The VGG16 model would also serve to us a benchmark model with which will be comparing the metrics of our "light-weighted" model.

Evaluation Metrics

As mentioned above, the dataset provided is slightly imbalanced which means that the accuracy as a metric shall be used with caution, if used at all.

The job of the model is to classify images, which technically means that the model should assign a probability of each image belonging to a certain class. Here is an example of the VGG16 model output when calling predict method on an image of a Scotch Terrier. The value of 0.184177 is a result of a likelihood function, which is the probability of the label that was actually observed.

Calculating the log of the likelihood of all actual labels and multiplying them by “-1” gives log loss – the metric which should show in probabilistic terms how close the prediction actually is to the true label.



```
[('n02097298', 'Scotch_terrier', 0.184177),
 ('n02102973', 'Irish_water_spaniel', 0.13394676),
 ('n02105412', 'kelpie', 0.11665777),
 ('n02106382', 'Bouvier_des_Flandres', 0.041713491),
 ('n02113799', 'standard_poodle', 0.039206326)]]
```

For instance if the actual label of the output above was “Irish Water Spaniel”, the accuracy of that particular datapoint would be 0. However the log loss would be 2.04 and capture the fact that the second best prediction was actually correct.