Machine Learning Engineer Nanodegree
Capstone Project
Ilia Panshenskov
August 8, 2021

I. Definition

Project Overview

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). Specifically in this project we are looking into the problem of classifying dog breeds from the input image of a dog.

Problem Statement

The problem on hand concerns training a computer algorithm in a way that given an image of a dog it can predict with a high accuracy what breed that dog belongs to – Doberman, Siberian Husky, Shar Pei, Labrador Retriever etc. The expected output should include a list of breeds that dog belongs to ranked by probability (i.e. certainty).

One of the key challenges of this problem is processing power. Working with image data implies a significantly larger size of data than, for example, in case of transactional or user behavioral data. Every image is encoded in a set of pixels, which in turn is encoded by a spectrum of colors. That implies thousands of data points for each image.

Thus, exploring and processing this sort of dataset requires significant and flexible computing capabilities. One notion that was discovered in the course of this project is that while GPU might be very efficient when it comes to training large sets of imagery data, CPU might outperform GPU when given a task of manipulating smaller datasets.

When attempting to parse image data into a multidimensional array the GPU got stalled both in Udacity Workspace and Google Colab, while the same task was completed on local CPU roughly within 2 minutes. Training a Convolutional Neural Network with over 27,000 parameters however is not a job that CPU could take up. Therefore in the course of the project we combined the usage of CPU and GPU the exact process of which will be outlined below.

Metrics

In the field of machine learning, accuracy is most usually a metric generally used to measure how successful the model is and defined by the equation:

accuracy = (true positives + true negatives)/sample size

Although we intend to observe this metric, it can be certainly biased when it comes to an imbalanced dataset. The metric that captures not only the class prediction but also the probability of a certain data point belonging to certain class is log loss (i.e. binary cross-entropy).

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Put in simpler terms, log loss is calculating log of the likelihood of all actual labels and multiplying them by "-1". For example, here is the likelihood that the pre-trained VGG16 model assigns to each breed:

```
[('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)]]
```

Given that the "Scotch Terrier" is the actual label for the image given, we would need to take a logarithm of 0.18 and multiply it by "-1" to get the final log loss.

We are focused on this particular metric because it gives a better feedback of what the algorithm makes of an image. It is also informative when exploring the false labels that the model has assigned. For example, the dog in the image above could indeed look like an Irish water spaniel or Kelpie, so even a a person would not be able to recognize the difference.

II. Analysis

Data Exploration

For the task on hand, we will be using the dataset compiled by Stanford's Computer Vision lab. It consists of 20,580 color images of dogs (available for public access

on [the lab's website](). There are 120 different breeds of dogs, ranging from very rare such as Tibetan Mastiff to rather common, for example – Boxer.

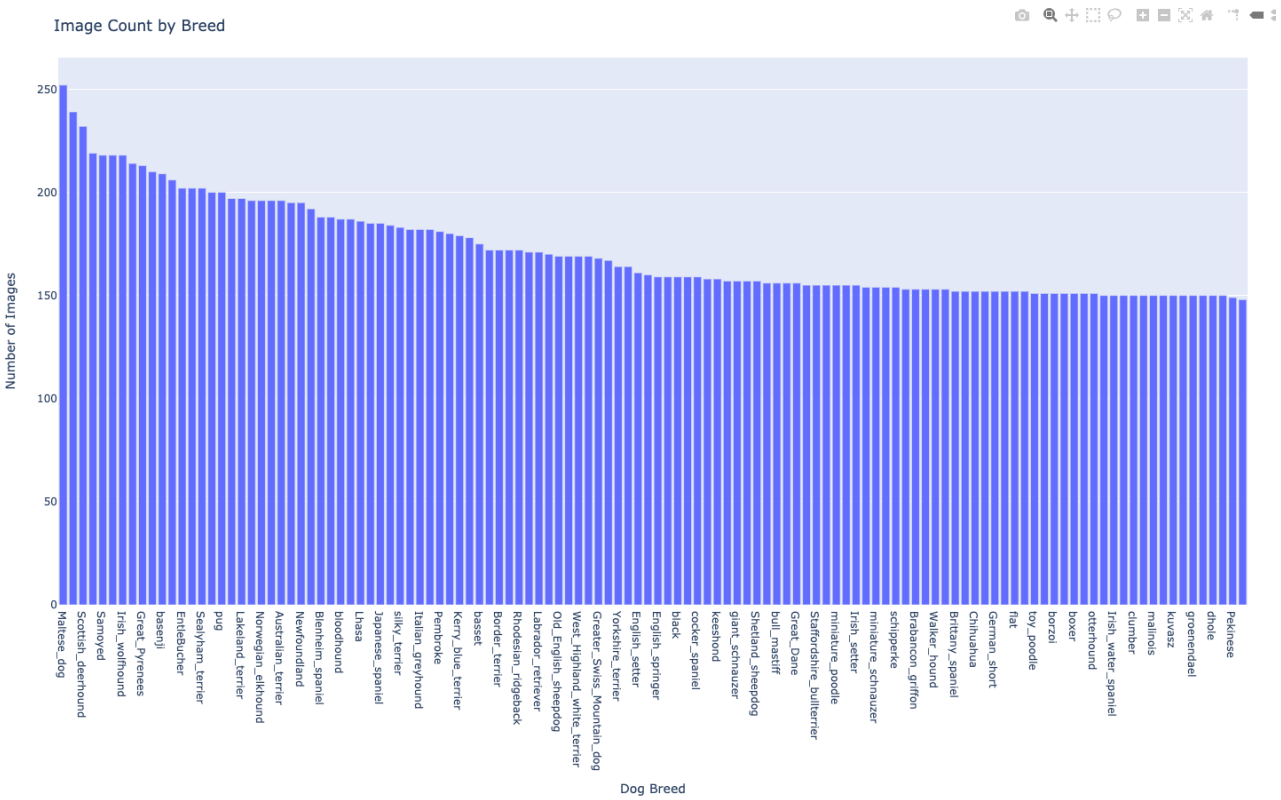
While some of the images are taken with a clear and contrasted background, the others are taken in a way that might be challenging for a model to decipher. Here are the three images of a Korgi that are present in the dataset. While picture A has the dog in a clear outline, in picture B the dog almost blends with a surrounding environment. Picture C has a similar problem as picture B, but also there is a number of other objects in the photo including a person. While we expect that the algorithm will detect a Korgi in image A, image B and C present a problem of a different caliber.

Image A

Image B

Image C

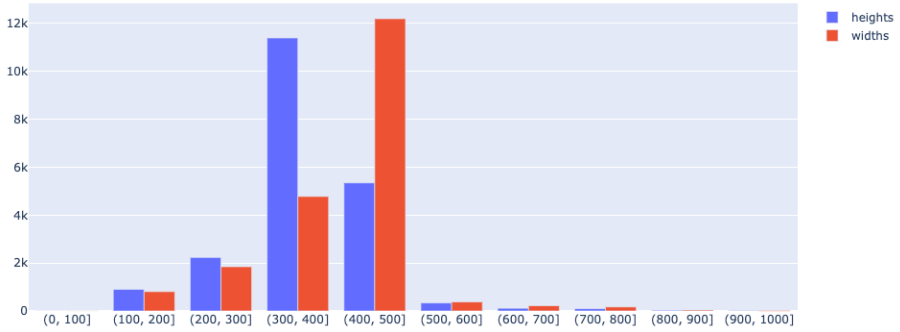
Additional characteristic of the dataset is that it is slightly imbalanced, which means that there are more classes of certain breeds than the other as can be seen on the chart below:



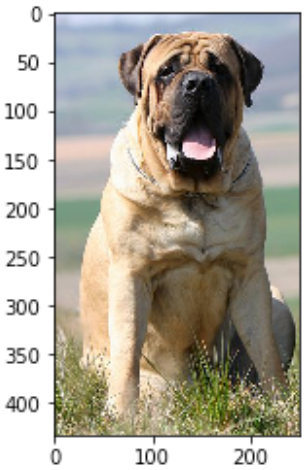
For example, there are almost three times as many images of Alaskan Malamute as of Xoloitzcuintli. This skewness of data made us choose not only accuracy as one of the metrics, but also log loss.

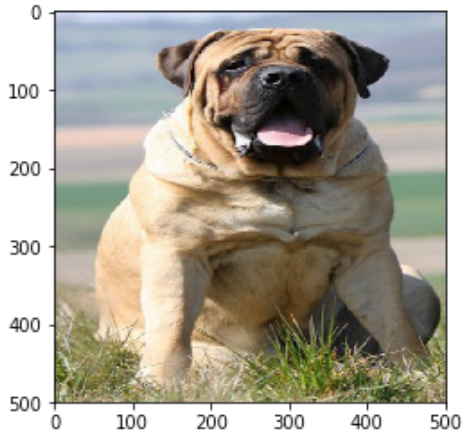
Analyzing the sizes of images revealed a large variance of widths and heights, the distribution of both skewed to the right with a median being around 385 for height and 442 for width.

	img_height
count	20580.000000
mean	385.861224
std	124.863446
min	100.000000
25%	333.000000
50%	375.000000
75%	453.000000
max	2562.000000



It means that ideally, when processing images we should normalize each picture to the size of 400 x 400 since the CNN model expects all data points to have the same shape. 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.





That being said, since the final approach that we will be leveraging to solve the problem is transfer learning with a help of a pre-trained VGG16 model, we will resize all images to 224x224, as this is the format the model requires.

Having pointed out the difficulties that we outline in the Problem Statement regarding computing capacity and the intricacies of the dataset we intend to make the following **major concession to the ideal approach:**

We will take the top 59 most common breeds of dogs whereby the image count for each breed is more than 160. Supposedly, this way the model should be better at understanding the characteristics of each breed and classify more accurately compared to a bigger challenge of identifying among 120 classes. The number of breeds was chosen arbitrarily, but is more than a half of the dataset.

Since the ultimate goal of this project is to establish a prototype and learn, we are willing to make this concession. Furthermore, when attempting to train the model on color images with CPU we observed that the log loss and accuracy of the first few epochs was not significantly higher than when trained with the above concession. This might imply that the trade off of accuracy and expense is rather reasonable.

Algorithms and Techniques

Image processing in this project was done fully in Python enironment with support of such packages as Numpy and Pandas. For the purpose of image display we used PIL package and submodules of Keras.

Image classification was done by employing Convolutional Neural Network architecture implemented exclusively in high-level Keras API. The architecture of the model iwas inspired by the VGG16 model's architecture, outlined in 2015 Simonyan and Zisserman, which we later one used for transfer learning. The VGG16 received its name after the concept of a "very deep Convolutional Network" that consists of 16 layers. The most outstanding aspect 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.

As will be laid out below, we attempted to build and train a custom model trained on a limited dataset of dog images. Although our model outperformed random guessing by a factor of 10, it was far from the perfect accuracy, therefore we decided to use transfer learning with a help of pre-trained VGG16 model.

III. Methodology

Data Preprocessing

Revisiting the matter of GPU and CPU and the method that we employed to work with the computing power issue, it is worth describing how we switched between processing capacities and how data was parsed.

The images for the dataset were stored in the directory on the local machine, so the steps of retrieving and processing data included several steps:

	file_path	image_id	breed_label
0	/Users/ilya/Desktop/U-Capstone/Images/n0209765...	n02097658_26.jpg	silky_terrier
1	/Users/ilya/Desktop/U-Capstone/Images/n0209765...	n02097658_4869.jpg	silky_terrier
2	/Users/ilya/Desktop/U-Capstone/Images/n0209765...	n02097658_595.jpg	silky_terrier
3	/Users/ilya/Desktop/U-Capstone/Images/n0209765...	n02097658_9222.jpg	silky_terrier
4	/Users/ilya/Desktop/U-Capstone/Images/n0209765...	n02097658_422.jpg	silky_terrier
...	...	...	...
20575	/Users/ilya/Desktop/U-Capstone/Images/n0208907...	n02089078_237.jpg	black
20576	/Users/ilya/Desktop/U-Capstone/Images/n0208907...	n02089078_222.jpg	black
20577	/Users/ilya/Desktop/U-Capstone/Images/n0208907...	n02089078_1021.jpg	black
20578	/Users/ilya/Desktop/U-Capstone/Images/n0208907...	n02089078_183.jpg	black
20579	/Users/ilya/Desktop/U-Capstone/Images/n0208907...	n02089078_2110.jpg	black

20580 rows × 5 columns

1. Parsing through the directory with the help of Python's OS package that allowed to navigate between folders and manipulate files. This gave us a structured pandas dataframe with top level metadata about the pictures (as presented in the screenshot above). The rows of the dataframe were shuffled and the dataframe was then split into train and test dataframes (80/20 split).

2. In order to turn images into numerical data representing pixels and the mix of RGB shades we implemented the function which opens the image through the file path, resizes the image and finally presenting it as a multidimensional numpy array.

3. The most computationally expensive step of data processing leveraged the function described in step#2, which was used in a loop that iterated through each row of data in the dataframe that we put together in step 1 – separately for train and test dataframes. The labels representing which breed the image belonged to were consequently hot encoded.

Finally the datasets were reshaped as numpy arrays: (8981, 224, 224, 3) for train and (2245, 224, 224, 3) for test dataset. The shape of the array encodes four important properties of the datasets: first refers to the number of images, second and third – size of the image, and fourth – the RGB mix of the given pixel.

Implementation

In this section we will elaborate on the algorithms and techniques that we used in the first approach when attempting to build a custom model.

The main tool of classifying images in our project was Convolutional Neural Network, that is – a neural network with a set of convolutional layers. On that note is it worth mentioning the building blocks that our network consisted of:

– **Dense layer** of the neural network receives input from all units of the previous layer (either input layer or any other neural net layer) and passes the values after to the next layer after performing activation function. Essentially, this is matrix-vector multiplication.

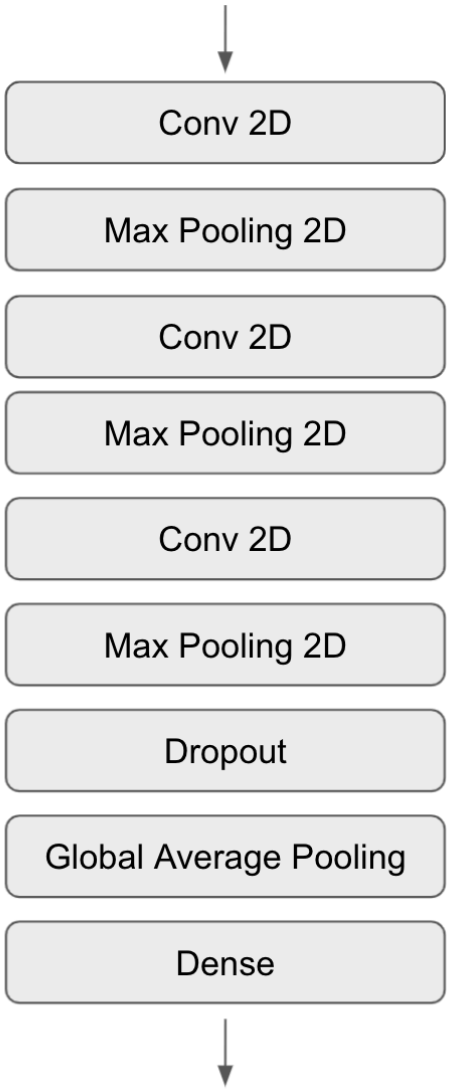
– **Convolution** is a mathematical operation whereby a small matrix of weights (i.e. a kernel or filter) "slides" over the input two dimensional matrix performing an element-wise multiplication with the part of the input and summing up the results into a single

output pixel. Thus, this operation derives the most distinct features of the image, making classification easier.

The two main techniques that go along with Convolution layer are **padding**, which allows to capture all image pixels including the edges by creating an additional contour of pixels, and **striding –** adjusting the "step" that the kernel takes between input pixels.

The downside of the convolutional layer is that it generates a large number of parameters, which makes it prone to overfitting.

– **MaxPooling** is a simple operation which summarizes a group of pixels based on its maximal value. The key feature of this technique is a "window size" – the perimeter of pixels which should be used to output a maximum value.



– **Dropout** is a strategy designed to prevent overfitting whereby a random subset of neurons (units) is dropped, i.e. not passed to the next layer of the network.[1] This approach allows to train on different parts of the dataset.

– **BatchNormalization** is another method for preventing overfitting. It takes the values given by a particular layer and rescales it with a mean being 0 and standard deviation equal to 1.[2]

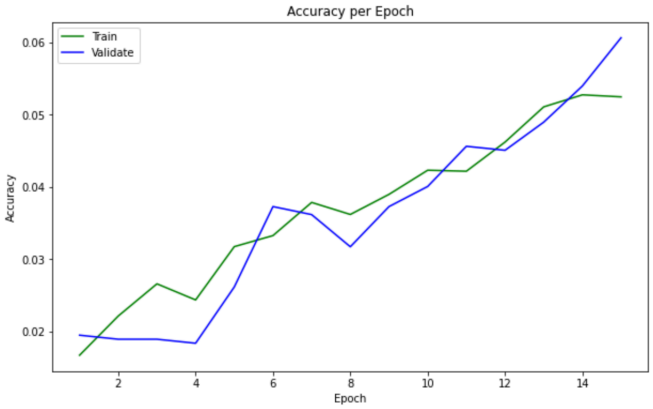
A good architecture of the neural network implies a robust net of convolutional layers that are reasonably regularized by a set of techniques that prevent overfitting so that the model would be able not to only learn, but also predict unseen input well.

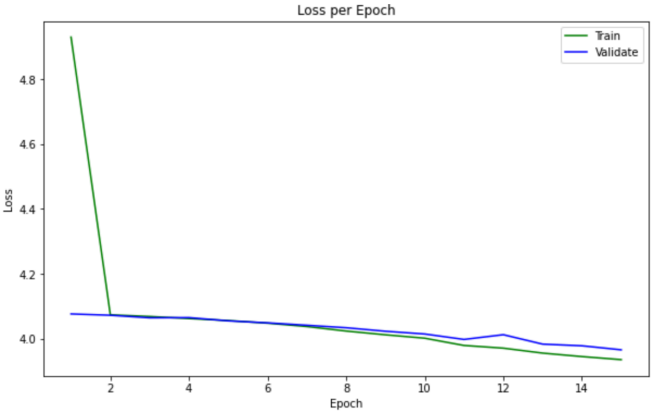
In our particular case we initially used a sequence of convolution layers, broken apart by regularization layers as depicted in the diagram here. This is the **benchmark model** which we started out with.

Thanks to regularization layers, we brought down the total number of parameters to 7,035.

[1] https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[2] https://arxiv.org/abs/1502.03167





We chose to optimize the training process with Adam algorithm, set the batch size for 50, directed 20% of the dataset to validation and trained the model for 15 epochs. The following two charts display train and validation losses plotted against an epoch and test and validation accuracies plotted against an epoch.

On the one hand, it is quite clear that the model does not overfit and progresses as the epochs go on. On the other hand, both metrics are far from the ideal result. Particularly, log loss showed an interesting trend whereby the loss dropped after the first epoch, but slowed its progress in the next epochs. **Result of the benchmark:** log loss of 3.96 and accuracy of 0.03.

Refinement of the Custom-Built Model

In the course of the refinement of the model we focused on several different aspects:

First of all, the charts of training and validation did not explicitly indicate the right number of epochs that we should train the model on. Therefore we introduced a built-in Keras method "early stopping" ("optimizing the optimization") and set a rather high number of epochs. Although extremely slowly, the algorithm yielded better log loss and accuracy after 27 epochs and then plateaued. Result: log loss of 3.9 and accuracy of 0.055 on the test subset.

Next, we wanted to imitate what was achieved with the development of the VGG16 model and add additional layers of convolution so the model would learn more complex features. We also added one more layer of max pooling (pool size = 4) after each convolution for regularization purposes. This small iteration yielded significant improvement to the model metrics on the test subset: **log loss of 3.8 and accuracy of 0.092 on the test subset.**

Interestingly, although adding two more convolutional layer brought a significant improvement to the model's accuracy, the early stopping callback interrupted the model training after 10 epochs, which implies that if we were to train the model longer, it would lead to certain overfitting.

To counterweigh the impact of additional layers, we decided to introduce modifications to the regularizaiton layers – dropout and max pooling, as well as adding batch normalization. In combination with this layers the model reached the log loss of 3.54 and accuracy of 0.16.

Implementing Transfer Learning

It is quite clear that no matter how sophisticated our custom-built model is, the training dataset is too small to make the model learn the features well. Hence, we decided to switch to the method of transfer learning.

The concept behind transfer learning implies reusing the model weights that were developed when training the model on large image datasets. In simple terms, in transfer learning the pre-trained model comes with the inherit knowledge of certain features, thus making the prediction better.

Upon loading the VGG16 model with Keras high-level API we have frozen the layers that came with the trained model, thus making 25 million parameters non-trainable. This was done in order to preserve the model's knowledge that was gained in the course of original training on the large dataset.

In order to make the VGG16 model able to train on our dataset we added one Flatten layer followed by two Dense layers with output of 59 nodes aligning it with the number of classes.

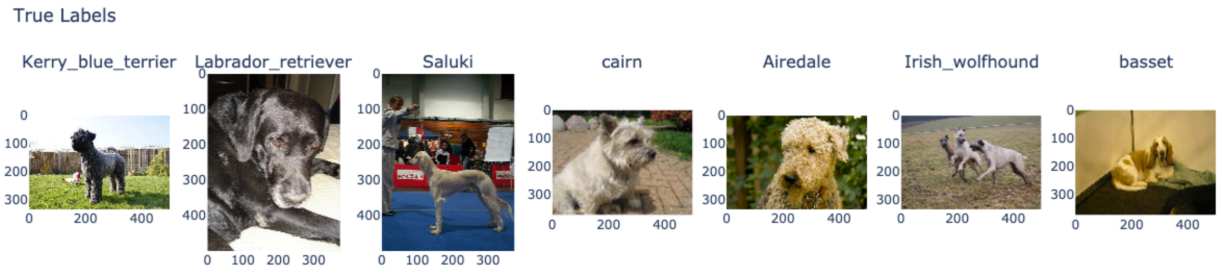
The technique of early stoping has proven to be extra useful in the training if the custom-built model, therefore we employed it in the process of training VGG16 model as well. As a result, model reached the accuracy of 39% and log loss of 3.49 just after 5 epochs.

IV. Results

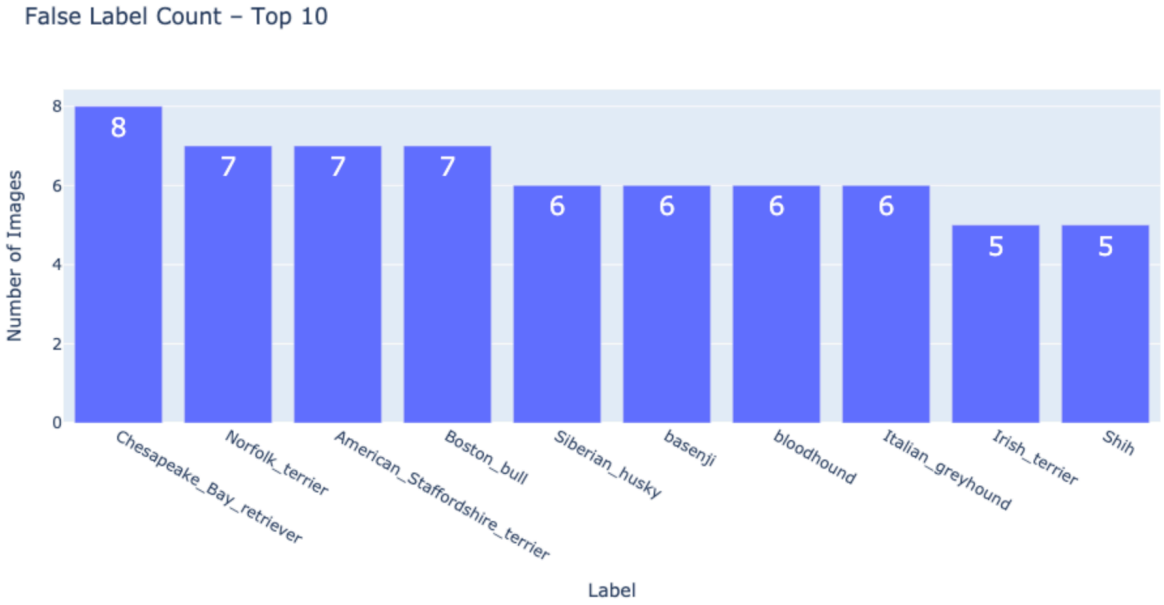
Model Evaluation and Validation

The final iteration of the custom-built model brought us the log loss of 3.54 and accuracy of 0.16. Although this is still a rough and and unusable result, we increased the accuracy of the model by more than 500% (from 0.03 to 0.16). This was arguably achieved by adding additional CNN layers and regularization framework.

The final version of the model developed through transfer learning accuracy is 36% and log loss of 4.03 on the test subset. Here are the examples of how the model identified breeds of given dogs in the test subset:

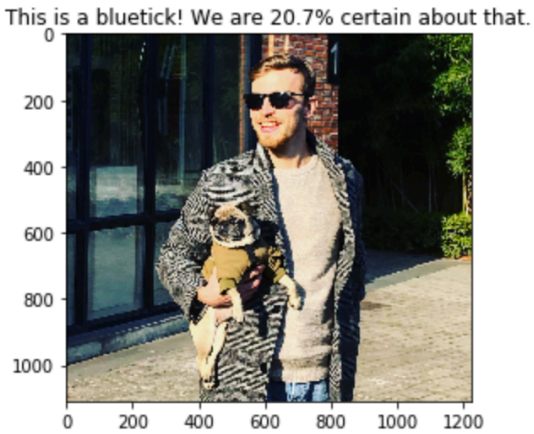


We also analyzed at which particular images the model fails to identify the breed correctly. Although this chart exemplifies some skewness towards some breeds that



are harder to decipher, our educated guess is that it is not about the breed per se, but about the way the image was taken (see the complexity that we described in the "Data Exploration" section).

The final test of the model was to see how well it generalizes to unseen data. We used the image of the pug named Dumpling (the image on the left) – a rather easy test since a pug has a set of noticeable features ("alien eyes", "flat nose" and peculiar creases of skin) and is clearly the dominant object in the image. Indeed the model



unmistakably recognized Dumpling as a pug. However, putting the model "under stress test" and requesting to make a prediction on the image of a pug in a hand of a person (image on the right) with unclear background made it return completely wrong prediction.

Thus, we would wager that although the model performs reasonably well on a carefully cropped and prepared images, it fails to identify the dog breed in a noisy environment.

Reflection

In this project we developed a machine learning model that was able to identify the breed of a dog from an image input.

The problem solution consists of three parts and logically presented in three separate Jupyter notebooks:

1. Data Pre-processing, done on local machine with CPU support the output of each is a set of 4 numpy files, that was imported to Python environment in Google Colaboratory in step 2.

2. Model Training was fully done in Google Colaboratory with GPU support and has to sections:
 – development of custom-built CNN model
 – development and training of VGG16 model (transfer learning)

3. Model Evaluation – evaluation of the "pickled' CNN model on the test subset and consequent application on individual images. Done on the local machine with CPU support.

The biggest challenge of this project was processing imagery data, which resulted in the need to switch between CPU and GPU, thus "juggling" datasets between environments. One major learning from the project is that while GPU might be reasonable and appropriate in training the model, it does not allow for processing rather small datasets.

Regarding the CNN model architecture itself, it was rather interesting to observe how accuracy metrics change dramatically event after adding a couple of layers. Furthermore, it was quite disarming to see how well the approach of transfer learning worked compared to the benchmark. We could have as well reverted the process and started the project with VGG16.

Improvement

As with the rest of machine learning projects and that is what the biggest experts in the industry reiterate again and again: the input data makes most of the model success.

The issues that we pointed out in the Data Exploration section were, no doubt, the biggest impediment to the model accuracy. For tackling this problem better we would imagine a machine learning pipeline of three stages. The first stage should identify locations (bound boxes) in the given image and crop them. Then these cropped images are passed to the second stage which is responsible for identifying the object: “person”, “chair”, “mug”,”dog” etc. Dog images are passed to the third stage which does exactly what we developed in the project: identifying the breed of the dog.

Additional tuning could be done with the architecture of the Convolutional Neural Network – tinkering with the strides and padding of the convolutional layer, changing pool size and switching optimization algorithm could make a difference. However, we remain skeptical about the significance of such changes.