June, 2021

Classifying Dog Breeds

with Keras

By

Abe Eyman Casey

and

Sameer Patel

Using a Convolutional Neural Network (CNN) built with Keras to identify dog breeds based on input photos.

# Introduction

The intent of this analysis is to use neural networks, specifically convolution neural networks (CNN), as a foray into applying machine learning methodologies to real-world applications, most prominently image classification.

The plan of approach is to use a CNN for this image classification task, because it is both the simplest and best machine learning tool to accomplish this kind of problem. To accomplish this, we will use the keras framework built in python.

This exploration will consist of the following:

* Loading the data
* Exploratory data analysis (EDA)
* Data preprocessing
* Model building
  + Keras Convolutional Neural Network
* Model evaluation / visualizations
* Conclusion / future analyses

The conclusion will assess the efficacy of the trained neural network based on the metrics returned in the model evaluation phase, as well as suggest future routes of exploration related to this problem.

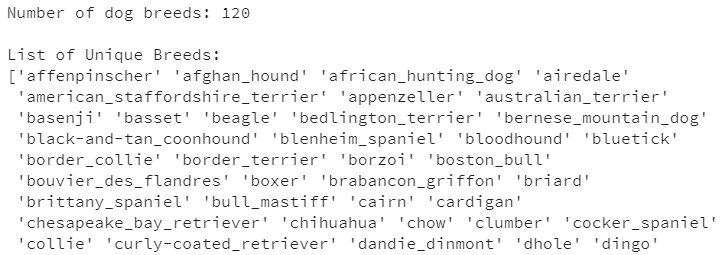
# Dataset Description

Data is sourced from <https://www.kaggle.com/c/dog-breed-identification/overview>.

The data set contains a training set and a test set of images of dogs. For this exploration, we will exclusively use the training set (train.zip) but will split it in order to obtain a labeled test set.

The training set contains over 10,000 images split into 120 breeds of dogs.

Below is a snippet of the labeled dataframe, consisting of two columns (one for the image ID, and another for the pictured breed). Adjacent to the dataframe is a snippet of the unique dog breeds contained within the training dataset.

# Data Preparation

The only features within this dataset are the image ID and the accompanying label.

Below is a small sample showcasing the range of image types (e.g. pictures with multiple dogs, varying levels of zoom/blur, other animals, etc.).

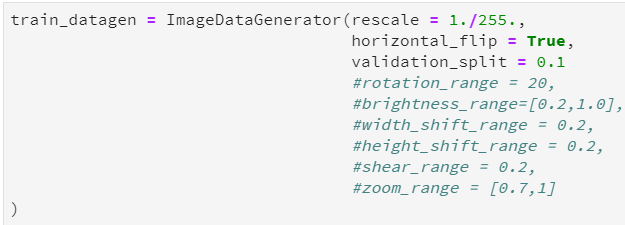


The wide range of image types suggests a need to augment the image batch prior to model fitting. We will do this by using the ImageDataGenerator class from the keras library to perform data augmentation on the images in the dataset.

The generator applies a series of random transformations to each image in the batch and replaces the training images with a set of new images for future usage in model training.

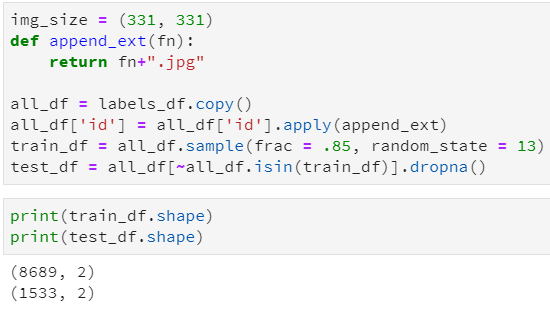
Several transformations are possible using the ImageDataGenerator function, but because of the large number of images already present in the training dataset, we opted to only apply the horizontal\_flip() transformation to augment the training data.

More augmentations would theoretically increase the accuracy of the model for predicting unlabeled images, but the additional computational cost is not justifiable for the purposes of this exploration.



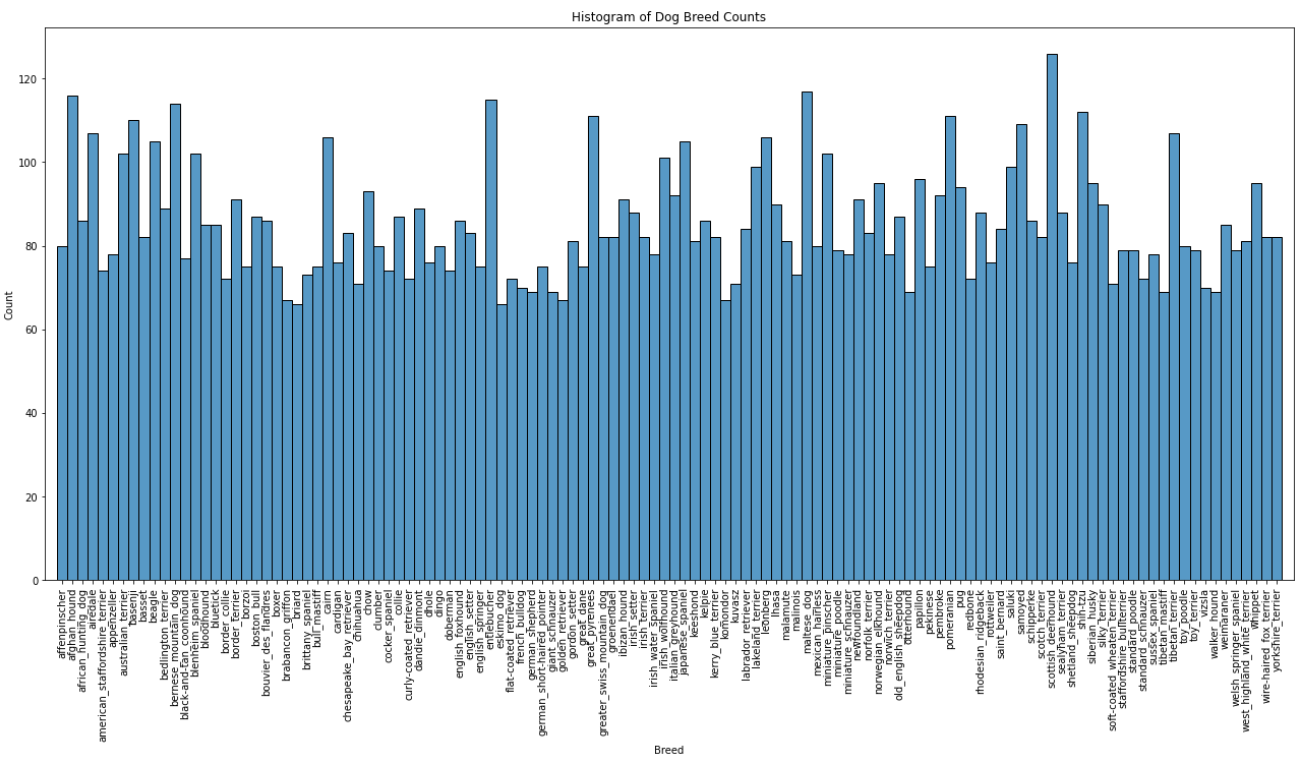
We will also split our dataset into train and test datasets, using a ratio of 85/15 train/test. Though the source contains a test dataset, the data is unlabeled, and for this exploration the decision was made to evaluate the model on labeled test data.

We split the labels\_df dataframe into training and test sets by utilizing the .sample() function in pandas.



# Data Analysis (EDA)

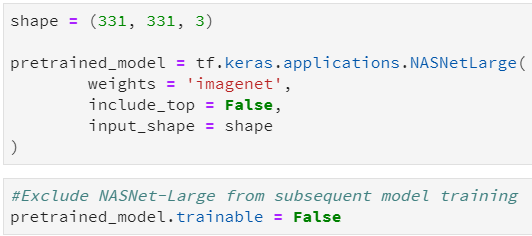
As aforementioned, because of the relative simplicity of the dataset, graphical options for EDA are limited. In this section, we will evaluate the class balance of the labels using a histogram of counts.



The histogram reveals a relatively balanced set of classes, ruling out the need for resampling prior to model fitting.

# Model Building

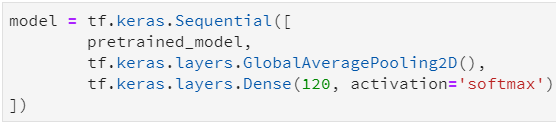
To build the CNN model using the keras library we added three distinct layers inside of a keras.Sequential model framework. For the base layer, we used a Transfer Learning approach which means we applied a pretrained model that was trained to solve a different but related problem which can assist our new model training and save an enormous amount of computational cost. This pretrained layer was the NASNet-Large which is a convolutional nearal network that was trained on more than a million images from the ImageNet database and helps classify many different objects. This layer of the model was essential in identifying which portion of the picture was the dog that was meant to be classified.



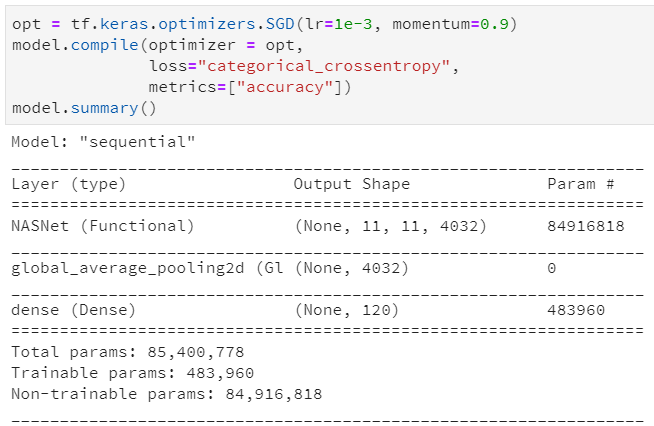
The keras library has a built in framework that allows for the NASNet-Large model to be used as a pretrained model as seen above.

The next two layers of the CNN model were a GlobalAveragePooling2D() layer and a Dense() layer. The first layer is a method of dimensionality reduction. Aside from global average pooling which reduces the applies average pooling on the spatial dimensions until each spatial dimension is one. There are other methods within the keras library for dimentionally reductive layers however the average pooling method is considered to be the cheapest method which was crucial given our computational limits.

Lastly, the Dense layer contained one node for each of the corresponding classes (120 dog breeds) and the activation = ‘softmax’ argument within that layer indicated that our prediction output would be a probability distribution of those 120 classes.



After building the layers the next step was to optimize the model. In this case we used a Stochastiic Gradient Descent (SGD) optimizer which we found to have the lowest memory requirements of the CNN optimizer approaches.



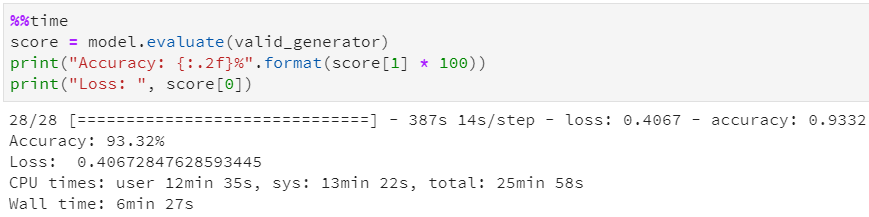
The increasing of the momentum argument accelerates the gradient descent in the relevant direction and dampens oscillations which can also reduce the cost of computation.

To calculate the step sizes, we took the proportion of the number of rows compared to the batch sizes for the training set and the validation set respectively. We fit the model on the train generator using 8 epochs. As seen below each epoch took approximately an hour to train on the machine.

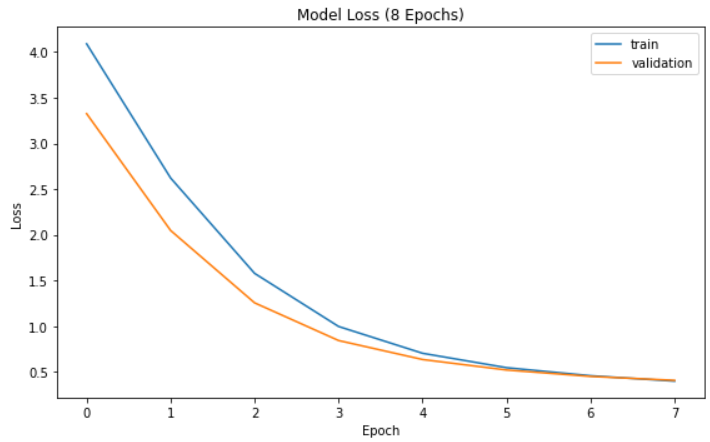


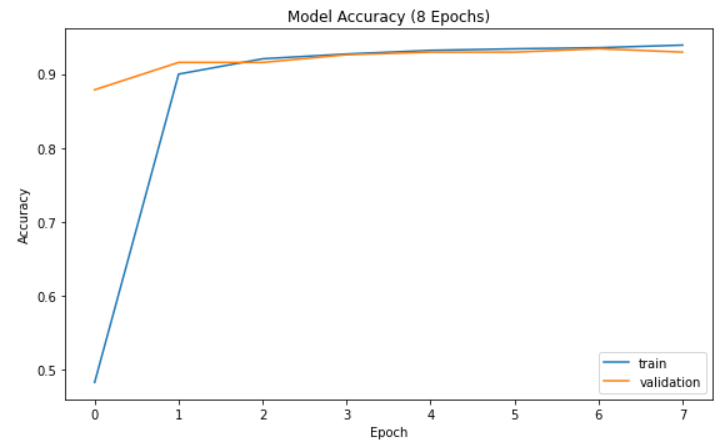
# Model Results

We will evaluate our model using the built-in evaluate() function, which returns both the loss and the accuracy of the final epoch.

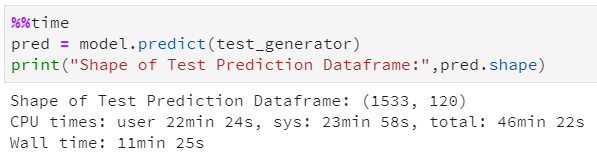


By visually inspecting the curves for model loss and model accuracy for the training and validation data, we can see that the metrics only marginally improve beyond the third epoch.





The model returned a loss of 0.4067 and an accuracy of 93.32%, which are acceptable metrics necessary to proceed with prediction on test data.





The model correctly predicted 1433 out of 1533 of the dog breeds within the unseen test image dataset, for an accuracy of 93.48%.

# Discussion, Conclusion, and Future Analysis

Given the large number of classes, the relative similarity in appearance between multiple dog breeds, and the variation of picture inputs, the above accuracy of 93.48% on unseen test data is an excellent result.

The visualizations presented above relating to loss and accuracy show that the CNN created in keras yielded an acceptable performance after around 3 epochs during training. This is most likely attributable to the usage of the NASNet-Large pretrained layer implemented in the model, which saved the model from having to retrain over 80 million parameters.

This foray merely scratches the surface of what is possible with CNNs.

Because of the cumbersome computational complexity and subsequent runtime associated with training the model, large-scale hyperparameter tuning was infeasible. Further routes of exploration relating to hyperparameter tuning include:

* Changing the ratio between the training and validation sets specified in the model initialization phase
* Adding to the data augmentation step implemented with the ImageDataGenerator class
* Changing the model blueprint itself vis a vis adding layers/nodes
* Uncountable others

As an attempt to bring value to theoretical stakeholders in this exploration, the team delved into creating a standalone webapp by which a user would upload a picture of their dog, and the trained model would yield a prediction as to its most-likely breed among the 120 breeds that the model had learned.

The unpublished application is pictured below after a sample trial picture was uploaded. The application returns the top 5 dog breeds predicted by the network, ranked by probability.

