Final Graduate Project: Image Classification

By: Emma Hashemi and Claire Medina

Abstract

In this image classifier, we created 15 features to encompass what we believed were the most important and notable categories for distinguishing animals in the given set of images. We created features to analyse shape and sizes, overall pixel and color intensities, and corner and edges in out photo set. After incorporating these features into different models such as: logistic regression, k-nearest neighbors, classification trees, random forests, and support vector machines, we analysed the accuracy of each model through cross validation and fine tuned them to create our final random forest classification model to yield acceptable accuracy ratings.

Introduction

The question being addressed in this project is nothing less than, what exact features and models allow for a classifier to properly conduct object detection in a given photo. In order to answer this question, one must obtain a working data set, for this project it was given, and then proceed to fully understand the format and usability of the data. From there, one must decide what parts of this data can be manipulated to create features that will be used for classification. This type of manipulation, of course, depends on the type and malleability of the data. After features and basic classification have taken place, the next step would be to tweak the classifier so that it yields the best results.

Description of Data

Given the nature of image data, it took many hours to fully access the data in a way that could be manipulated and transferred among jupyter notebooks. We began by composing the starting set of images into a dataframe consisting of one row per image containing the 3-d pixel array for the image as well as its specific encoded category (the type of object in the image). Each image object was further composed of a 3d array consisting of the rows and columns of pixels and their unique pixel red/blue/green makeup.

The notebook containing the starting and validation dataframes, originally resided in jupyter hub meaning that there existed a tight limit on the amount of memory access at any given time. During this phase, we created our first data frame and compiled it into a csv so that it could be accessed in Notebook II, yet when we opened it the next notebook, it had been transformed into a completely different format. It was then that we learned that because of the matrix like structure of our data, each new line created a ‘/n’ character in its given location. Following this

finding, we discovered a different way of formatting the data, namely as a pickle. This format however took up too much space in our jupyter hub memory to do at once, so we had to split up our starting data frame into multiple pieces to each be saved as picked and then recombined in the next notebook. In the middle of this process, we realized that performing all of these actions on our local devices would save an enormous amount of time and memory space, prompting our switch and solving many of the problems we faced up until this moment and preventing further memory shortages for future endeavors.

Methodology

Upon researching what qualities make an image unique, we found and decided to implement several features including pixel sizes, pixel intensities, any identifiable edges/contours and object corners. The first and simplestlest features included image size and image aspect ratio, as inspected by Figure 1 and 2. While there may not be extreme variance among the different coding categories, it is clear that certain animals on average do have different size images as well as aspect ratios. Infact, Figure 1 demonstrated to us that there are two clear patterns for aspect ratio which led us to create Figure 2. These features were then also used in the classifier.

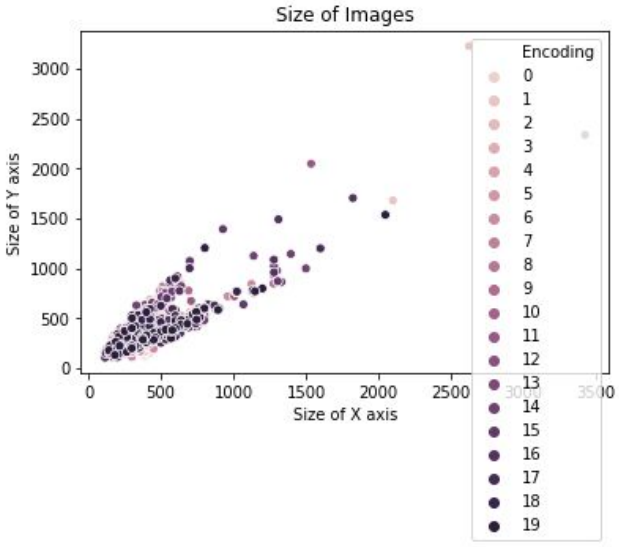

Figure 1

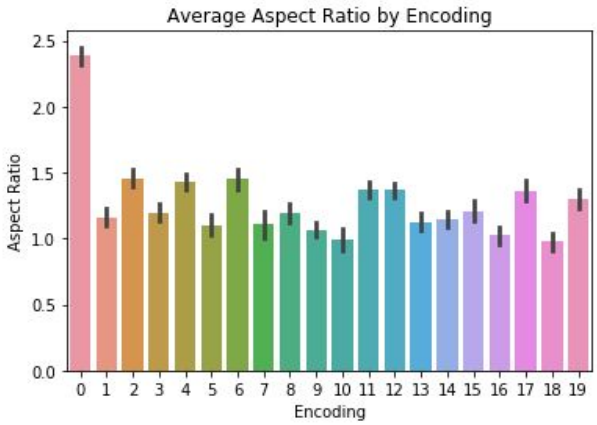

Figure 2

Color intensities

The next few features attempted to encapsulate the idea of pixel intensities. This would mean taking account for the amount of red, blue and green per photo. This prompted features 1, 3, and 4 which captured the average red, blue, and green image channels respectively. We soon realized that pixel intensity is not only captured by the amount of each color, there must be some way to note the actual power of the combination of colors that made up a pixel. The following graphs demonstrate the variance of rgb values per photo.

Image 1

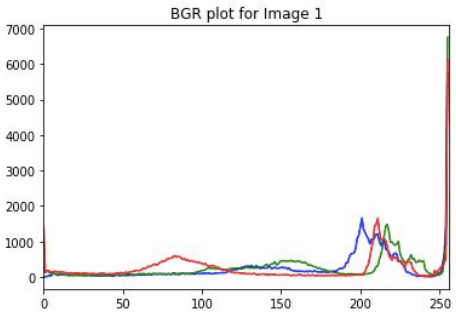

Figure 1

Image 2

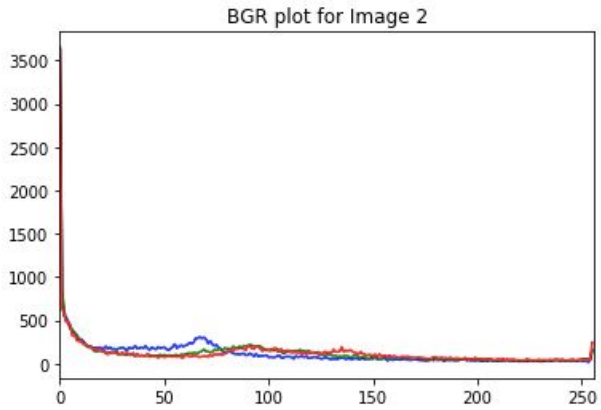

Figure 2

Image 3

Figure 3

By comparing the rgb color channel intensities, the overall color of the image should be captured into the feature embedding space. These color models do not however encompass overall pixel intensities. The muted green colors of Image 1 have higher average rgb values yet does not similarly compare the slightly more intense Image 3. For this reason we then created several features that attempted to capture the overall pixel intensity of the images. The next few features, features 12, 13, and 14 each contribute to the overall idea of image intensity. Feature 12 turns the photo into grayscale and then takes the average value of all the pixels in the photo. Feature 13 uses feature 12 to find the average value of the grayscale photo and then finds the average distance of each residual from the mean to try and capture the overall variance of pixel intensity, as illustrated in Figure 6.

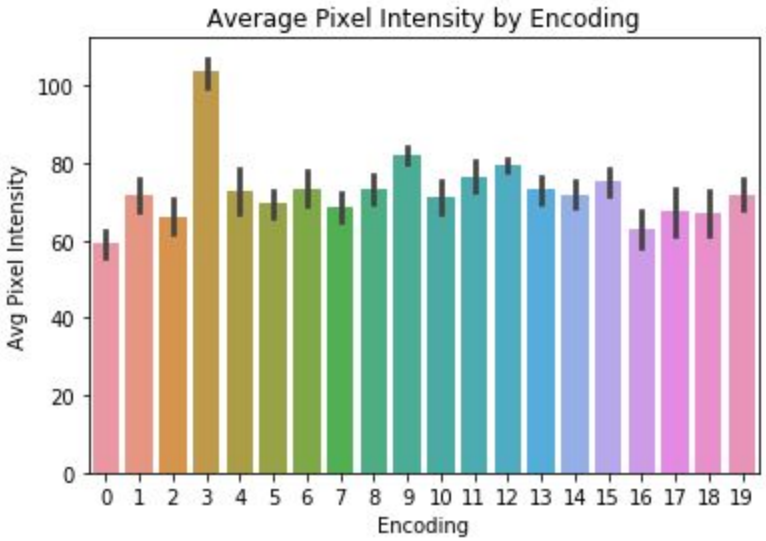

Figure 6

The final features created attempted to capture higher complex aspects of the photos such as the corners of objects or edges and contour detection. We realized too late that the output of these features did not comply with the necessary format of out classification models; nevertheless, we felt as though they were valuable enough to explain in this narrative.

Image 4

Image 5

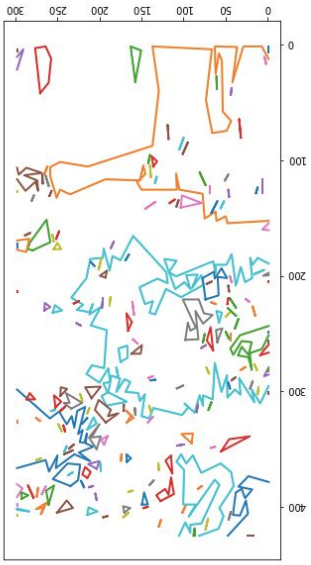

Figure 4

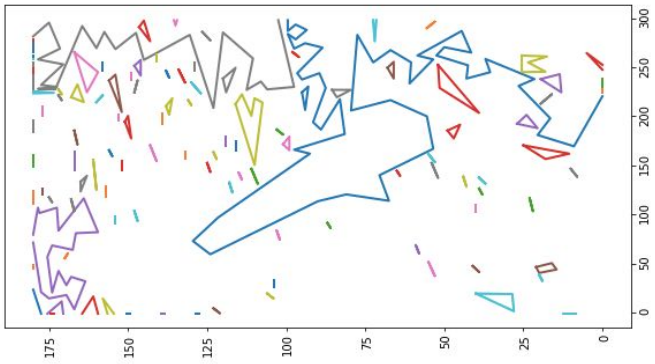

Figure 5

The above graphs illustrate the images created after running the original images through features 6/7. We were able to find source code that create the image array for the contours of the photos (Source 1). Since the resulting output was in array form of these edges, we decided to plot them to see if they actually resembles the actual images.

The final features created were meant to capture sets of important pixels, namely the corners of the images. We tried to use the “Harris Corner”, “FAST Algorithm for Corner Detection”, and “ORB (Oriented FAST and Rotated BRIEF)” libraries, yet they were not useful for our ending classifier (Source 2).

Model descriptions

When building our final classifier, we tested five different models: logistic regression, k-nearest neighbors, classification tree, random forests, and support vector machines. When using the logistic regression classifier, we found the training accuracy to be very low. When using the k-nearest neighbors classifier, we found the score of the model to be about 50%. The classification tree produced a model with around 40% accuracy. Both random forests, and support vector machines yielded scores that were close to 100% accuracy, which may be due to overfitting. Some of the models worked better than others because of the methods used in order to draw the boundaries for classification. However, some of the models may have done better compared to others due to overfitting.

For our final model, we chose the random forest classifier because it had a higher training accuracy and a lower misclassification rate on the validation set. However, we acknowledge that

due to our time constraint, we were unable to change the bias, noise, and other hyperparameters, when using our cross validation methods to find the best model.

Citations

Source 1:

Author: Mohammed Innat

email: innat1994@gmail.com

website: https://iphton.github.io/iphton.github.io/

Source 2:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_core/py_basic_opspy_basic_ops.html