DS 200 Grad Project Final Report

Ashwin Srikant, Kaushal Yagnik

7 December 2018

Introduction

In this project, we attempt to use manual feature engineering to classify images of primarily different types of animals. We extract scalar features from color images, and use a variety of machine learning models (SVM, Random Forest, K-Nearest Neighbors, and Logistic Regression) to create the best prediction model, as determined by performance on a validation set. Our dataset consists of color images of a few types of vehicles and a wider variety of animals. Although, this dataset is a relatively small dataset with a total of ~1500 images, the differences in color features could be useful in determining differences between them. Eventually, we also use TensorFlow to predict accuracies using a state-of-the-art deep learning technique.

Data cleaning and Preprocessing

We found ~16 black & white images in the training dataset, which we addressed by making the intensities of all the 3 channels equal to the intensity of the B/W image. We also resize all the images (both training and test sets) to a size of 128 x 128 pixels to have a coherent feature space.

Feature Engineering

Given the limitation of total 15-20 scalar features, the challenge was to find most informative features in that space. We felt that using intensity and color-based features would be better instead of going for features like detecting number of corners, since we would also need positional information for corner based features.

Color-Based Features

We hypothesize that different types of animals are likely to be seen in different environments and will therefore have different intensities in their color channels. For example, we'd expect to see a whale in water, so we'd expect a primarily blue landscape around it. This idea led us to the first set of features, where we look at the contribution that each color channel makes to the total landscape (measured in terms of % the number of times that color was the max intensity among all three channels). Our graphs also indicate that these are good features, as different animals display different levels of feature values.

We also take into account the contrast of the image, but we are surprised when the results showed that it does not have a significant effect on prediction accuracies. We think this could be because the dataset happens to be homogeneous in terms of contrast, which does not make it a great distinguishing feature.

Cropped-image Features

We felt the need to incorporate some spatial characteristic of the object to be detected since it would generally lie in the middle of the image. Hence, we computed the above features for a central patch of the image which may give us some information. Hence, we computed the color features specifically for the middle half of the image. Our ad-hoc analysis showed that there were some clear differences in values, which justify this approach.

Techniques from Computer Vision

Due to constraints on a limited feature space, we were stuck on how to best use information from computer vision algorithms that are trained to detect features. After initial experimentation with the few techniques like Harris corner detector, our best approach turned out to be the Canny edge detection algorithm. Instead of storing the actual edges, we store the number of edges (fraction of pixels with sharp intensity changes). Our rationale behind using this is that knowing number of edges would help in approximate the relative number of features, which might be helpful in distinguishing animals like a porcupine from a whale. This would definitely be a far from optimal approach, but in the absence of using vector features this could be one way to incorporate some computer vision based information.

One surprising thing was that while the Canny edge detector was useful as a feature, counting corners with Harris corner detector was not nearly as useful. We hypothesize that this is because the Harris corner detector is best suited for grayscale images, but we didn't want to lose the information gained through color.

Modeling

Since different ML models approach the prediction problem in different ways, it lead to differences in their performance. For example, K-nearest neighbors works best when samples are in a low dimensional space, because it's easier to quantify the distance between two samples, whereas an algorithm like Random Forests helps prevent overfitting by using an ensemble learner to make predictions.

In order to find the most optimal model over a range of hyper-parameter values, we performed extensive parameter tunings exploiting Ray parallelization to improve the run-times significantly. For each trained model, we evaluated it on a stratified cross-validated set to select the model having the best mean-validation accuracy amongst all the tunings.

The best model for us was turned to be Logistic Regression, with a validation accuracy of about 41%. We also had a few models in Random forest classifier which were giving around ~39% accuracy, but to be in line with the Occam's razor (less complex models should be preferred for overly complex models over miniscule gains in accuracy), we chose to go ahead with the Logistic Regression model. Based on the best performing hyper-parameters obtained as described in the process above, we used these parameters to train a model on the entire original training data to generate the final model to be used to make predictions on the final test data.

Installation Notes

1. To run the notebooks, please place the unzipped training data inside a folder called "data"
2. Notebook 2 uses cv2 package, so this needs to be installed to get it to run.
3. To run Notebook 3, please upgrade sklearn from v0.19 to v0.20.

Results

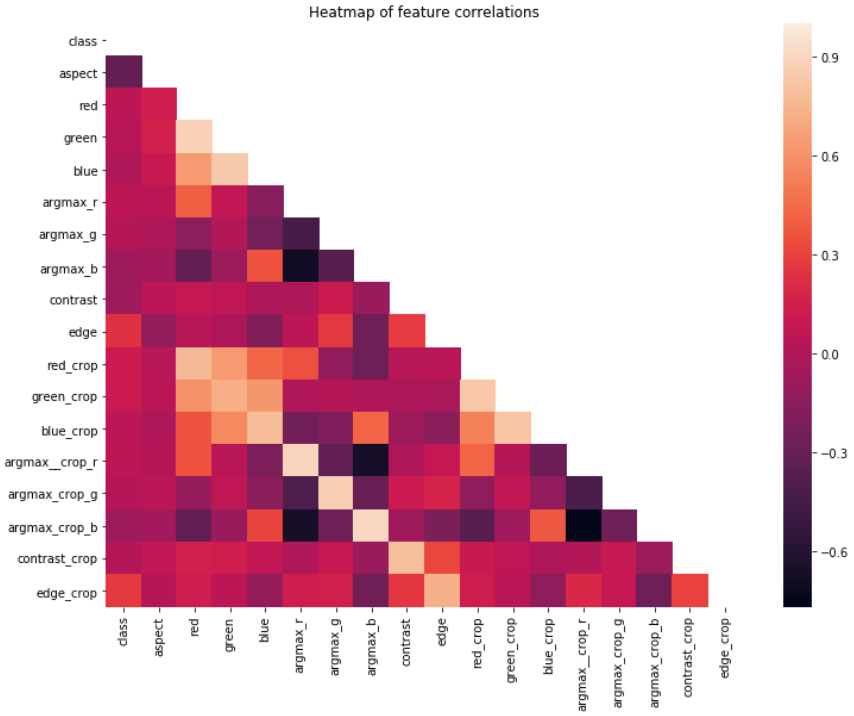

Figure 1: Feature Correlation Mapping

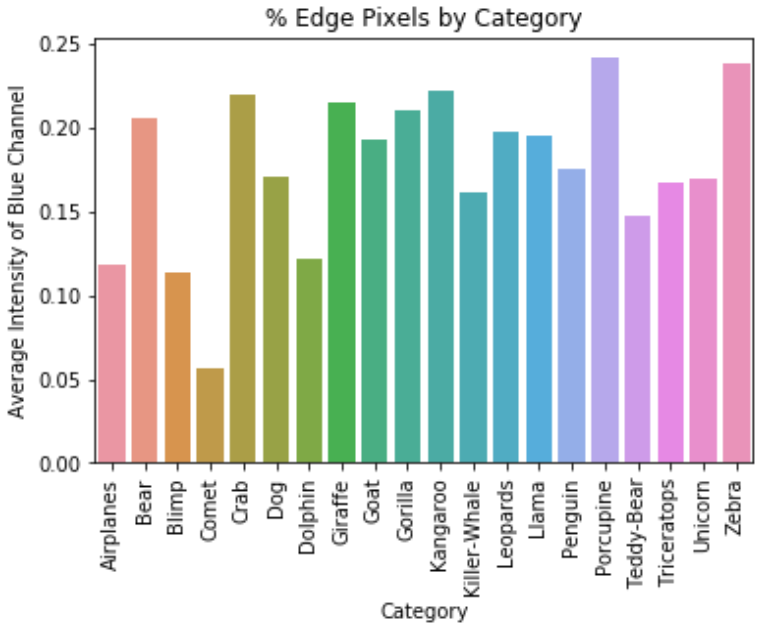

Figure 2: Edge Pixel Percentages Grouped by Category

The above plots show interesting features from our feature set (more detail is provided in the notebooks). The first plots like feature correlation, which we'd want to avoid between seemingly unrelated features. Luckily the features that have high correlation are ones we'd expect (red vs

blue channel) and others don't exhibit this pattern. The second plot looks at our edge counter data to show that different categories have different percentage of edge pixels, which is what we'd expect, making this a useful feature in our model.

Finally, we show the results of our ML models on our classification problem:

Classifier	Best Validation Accuracy (%)	Best Hyper-Parameters
Logistic Regression	~41.0	C = 50 (regularization)
K-Nearest Neighbors	31.5	K = 18 (neighbors)
Random Forest	36.5	400 trees, maxDepth = 5
Support Vector Machine	29.2	C = 81 (regularization)

Table 1: Machine Learning Model Validation Accuracies After Hyper-parameter Tuning

Conclusion

After selecting the features described above and running the ML models, we can see from above that the best validation accuracy achieved was about 41%. This seems low on the surface, but since we are predicting between 20 different categories, random chance would be approximately 5% so it is a stark improvement over that.

This project allowed us to explore object recognition and classification in a low-dimensional space, since we were limited by how many scalar features we could create. Going forward, if we were presented the same problem, other options for ideas we could try are: 1) Create more sophisticated features based on methods from computer vision (i.e. metrics similar to edge or corner detection but reduced to a scalar), 2) Use a Decision Tree-like approach in finding features that effectively split the category space evenly into two, so that within 20 features one could easily capture characteristics of each category effectively, and 3) Explore approaches like PCA and other clustering methods to get the high-dimensional nature of images down to a lower subspace.

Because we were categorizing primarily animals, another interesting approach would be to extend our color features by transforming the images into another color space, say HSI, and re-computing the features from above - many would change depending on how saturated the images were etc. That could be another avenue to explore low-dimensional image categorization although it would require more than 20 features.

Bibliography

1. "Feature Detection and Description¶." *Cascade Classification - OpenCV 2.4.13.7 Documentation*.
2. Nikishaev, Andrey. "Feature Extraction and Similar Image Search with OpenCV for Newbies." *Medium.com*, Medium, 15 Feb. 2018.
3. Sundriyal, Divyanshu. "A Simple Animal Classifier from Scratch Using Keras." *Medium.com*, Medium, 11 Apr. 2018.