Capstone Project

Machine Learning Engineer Nanodegree

Nathan George
July 28th, 2016

Definition

Project Overview

Americans love their dogs; in the US, there are about 80 million dogs[1]. In total, 500 million dogs are estimated to exist on planet Earth.[2] The American Kennel Club maintains a list of dog breeds, totaling 252 at the time of this report.[3] Dogs, like humans, have unique personality traits, and these can be generalized to the specific breed of a dog. It is important, therefore, that an owner's personality be somewhat matched to their dog(s). At the very least, it would be desirable for a dog owner to know what type of personality to expect from a dog before bringing one into their family—knowing a dog's personality traits makes training and caretaking of the dog much easier. This is easy enough to do if a dog is a purebred—simply ask the owner or breeder what type of dog it is. For hybrids, mutts, strays, or rescued dogs, determination of the breed becomes much more difficult.

Ideally, genetic testing can elucidate the breed, and hence personality characteristics and wellness requirements of a dog.[4] Unfortunately, this type of testing is not very cheap or fast, especially compared with taking a photo on a smartphone. Breed identification by photo would be nice—simply snap a picture with a smartphone app, and instantly get information on the breed(s) a dog could be. Genetic identification by photo would be useful for mutts or hybrid dogs, where the owner may not know what the genetic background of the dog is. This would also be useful for people looking to get a new dog—go to a dog park or walk around the neighborhood, interact with some dogs, and find their breed with your smartphone. More generally, identification of organisms by photo could be useful in the fields of botany and horticulture, cell culture, and agriculture.

Microsoft recently came out with an application using machine learning to identify dog breeds.[5] However, it can be hilariously incorrect—sometimes even the first Bing image result for a dog breed or the stock photo from akc.org of a certain breed is not correctly identified. Furthermore, they only offer an iPhone and web app, and the project is not open source.

The American Kennel Club (akc.org) maintains a list of over 250 breeds of dogs, along with information and pictures. This is a starting point for classifications and training data. Google and Bing provide access to the Internet's database of dog pictures, providing more training data. Wikipedia[6,7] also has a more comprehensive list of breeds than the AKC, since they include more international breeds. Wikipedia also provides a list of hybrid dog breeds.[8] Other online resources provide lists and pictures of dog breeds and

hybrids.[9,10] There are also some international Kennel Clubs that maintain lists of breeds.[11,12] In this report, the akc.org site and Google images have been mined for training and testing images.

Problem Statement

The goal of this project is to create a framework for an open-source machine learning application, which reports the dog breed that most closely resembles an input image. The 252 breeds listed on akc.org will be used to classify dog breeds by photo.

To create this application, training data must first be mined since a dedicated dog image database does not currently exist. Features need to be extracted from the training set; these could be texture, keypoints, and histograms of oriented gradients (HOG), and color information. A machine learning algorithm needs to then be trained and cross-validated on portions of the training set. The machine learning algorithms tested here will be support vector machine (SVM), k-nearest-neighbors (kNN), and random forest decision tree classifiers.

Metrics

Accuracy (fraction of correct predictions out of total predictions) will be used as a metric in this project. Recall, precision, and F1 scores would be difficult to use as a metric because there are so many classes (252), and the accuracy of the solution will likely be low (due to the large number of classes). The goal is to beat random guessing on the test dataset (used as the benchmark).

Analysis

Data Exploration

Intuitively, there are some features about pictures of dogs we use to discern their breed: color, fur texture, body shape, face shape, and size. Of these, we can easily calculate of all of these features except size. For this project, I chose to use fur texture as the main classification item, because the texture of dogs' coats tends to differ greatly between many breeds. Texture was measured with Haralick texture, which is a common way to compute texture of an object in computer vision.

I scraped the akc.org website for breed names and photos, as well as the first 40 images from Google image search for each breed. At the time of scraping, there were 252 breeds. In total I collected 10332 pictures that totaled around 1 GB in memory.

To better facilitate classification, I manually drew rectangles around the dogs and their faces in 1590 images. Using the cv2.grabCut() function, I separated the foreground (dog) from the background of the image, with the rectangle as a starting point for the k-Means algorithm behind grabCut. I hypothesized this would allow for better feature extraction

since the background noise would be reduced, and it would also allow for a comparison of the features of the dogs and the backgrounds.

I then used the grabCut mask to break up the foreground and background into rectangles, so Haralick features could be calculated for each. Haralick features were calculated for each 10px by 10px rectangle, and averaged over the foreground and background to get a single set of Haralick features for each foreground and background from each image.

Haralick texture outputs 14 features across 4 directions. Only the first 13 features are used in the mahotas Python implementation of Haralick (the 14th is considered to be unstable). The 4 directions are then averaged to arrive at a 13-dimension vector. Examples of the Haralick features stored in a pandas DataFrame follow:

breed: Manchester Terrier
filename: ef144fab7422d2cd75cea6f09c3936d2813e7ed4.png
foreground Haralick:
[8.57381478e-03 4.10863443e+02 5.87781055e-01 5.65823223e+02
1.21756636e-01 1.36821823e+02 1.85242945e+03 5.59116418e+00
7.13270498e+00 8.16455498e-04 4.44472415e+00 -6.28489862e-01
9.94489378e-01]
background Haralick:
[7.55733120e-03 2.25157047e+02 5.39230190e-01 2.34382991e+02
1.24854741e-01 2.01441970e+02 7.12374918e+02 5.45011728e+00
7.16052523e+00 4.89178675e-04 4.19504514e+00 -5.70747974e-01
9.96462007e-01]

breed: Belgian Malinois
filename: Belgian Malinois1-1.png
foreground Haralick:
[9.47353132e-03 1.18365183e+02 6.43529352e-01 2.25425319e+02
1.81155423e-01 1.46462244e+02 7.83336094e+02 5.26559912e+00
6.97106207e+00 1.13272036e-03 3.77818686e+00 -5.33584867e-01
9.91146700e-01]
background Haralick:
[8.51613072e-03 1.45234059e+02 4.15175619e-01 1.32440737e+02
1.51154189e-01 2.74604762e+02 3.84528888e+02 5.20807947e+00
7.04629907e+00 5.22782380e-04 4.01557325e+00 -5.10338264e-01
9.91503267e-01]

breed: American Water Spaniel
filename: American Water Spaniel1-3.png
foreground Haralick:
[7.71954248e-03 3.14988754e+02 6.55256903e-01 5.64596634e+02
1.17867088e-01 1.06535185e+02 1.94339778e+03 5.64032223e+00

7.17811101e+00 8.22199170e-04 4.42045798e+00 -6.32243356e-01
9.96467617e-01]

background Haralick:

[9.00257668e-03 1.83591413e+02 6.74463074e-01 2.76250367e+02
1.64121342e-01 2.74751251e+02 9.21410055e+02 5.44741620e+00
7.04161587e+00 5.33522373e-04 3.90348774e+00 -5.81688180e-01
9.93920011e-01]

Feature number 12 is negative, and the features span 4 orders of magnitude. The 2nd, 4th, 7th, and 10th Haralick features seem to differ the most in the chosen examples.

I reduced the dimensionality of the Haralick features to 3 using PCA. Then, using quartile outlier analysis, where outliers are said to be 1.5 times outside the interquartile range (Q3-Q1), I found 75 total outliers in the reduced Haralick features. I found 12 entries that were outliers in more than one dimension, and dropped these outliers from the dataset.

Exploratory Visualization

The Haralick features of the foreground (dog) seem to differ most for features 1, 2, 4, 5, 6, 7, and 10, as can be seen from a subset of the data:

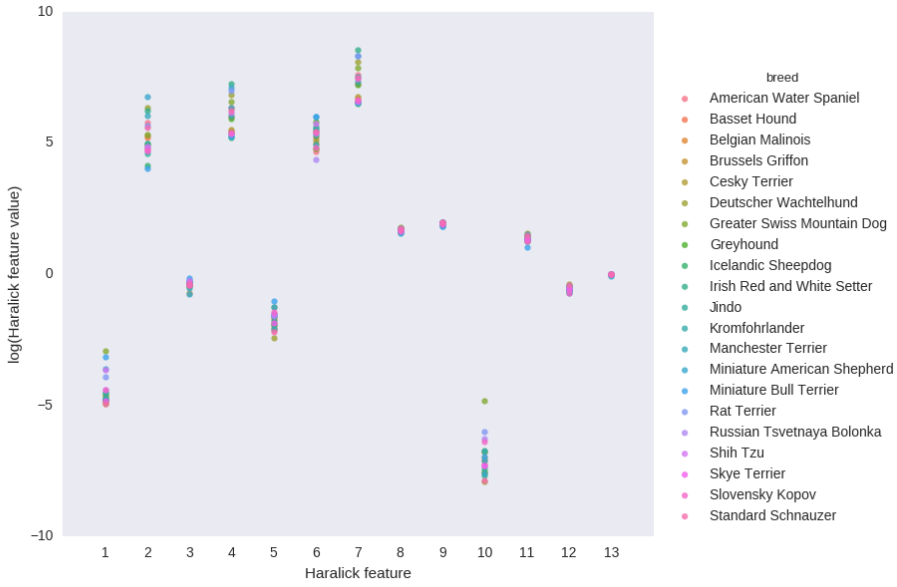

Figure 1: Distribution of the log(abs(x)) transformed foreground Haralick textures for a subset of dog breeds.

Overall, the standard deviations of the foreground Haralick features follow this trend:

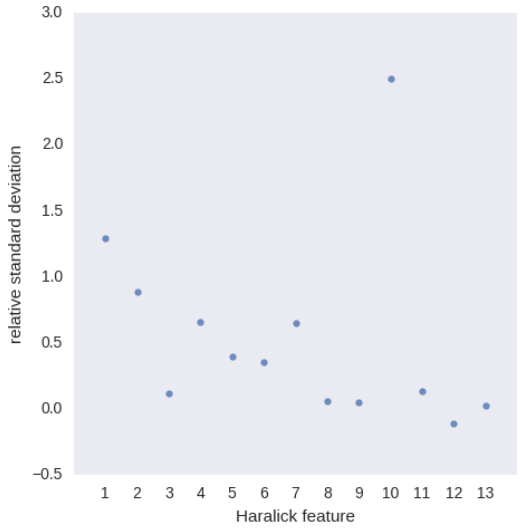

Figure 2: Relative standard deviation of the 13 dimensions of the foreground Haralick texture.

with feature # 10 being a clear outlier in standard deviation.

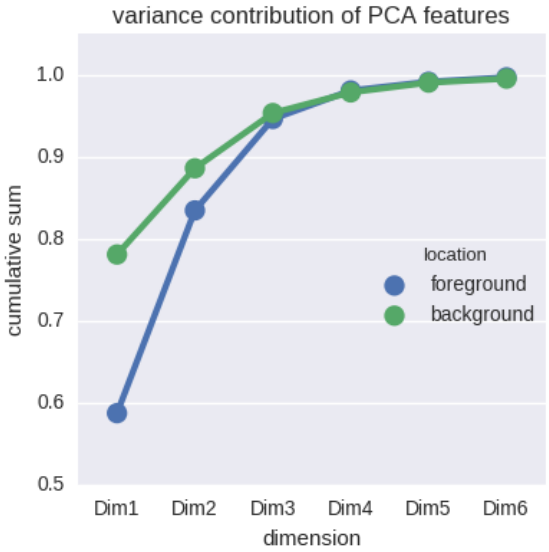

Figure 3: Cumulative sum of the variance captured by the first 6 Haralick PCA components.

Once the 13-dimension Haralick features were calculated, I transformed them by taking the log of the absolute value of each feature, since the features span multiple decades and some are negative. Principal component analysis was then performed; it was found the first 3 components of the PCA make up about 95% of the variation in all parts of the images, and the top 4 components about 98%.

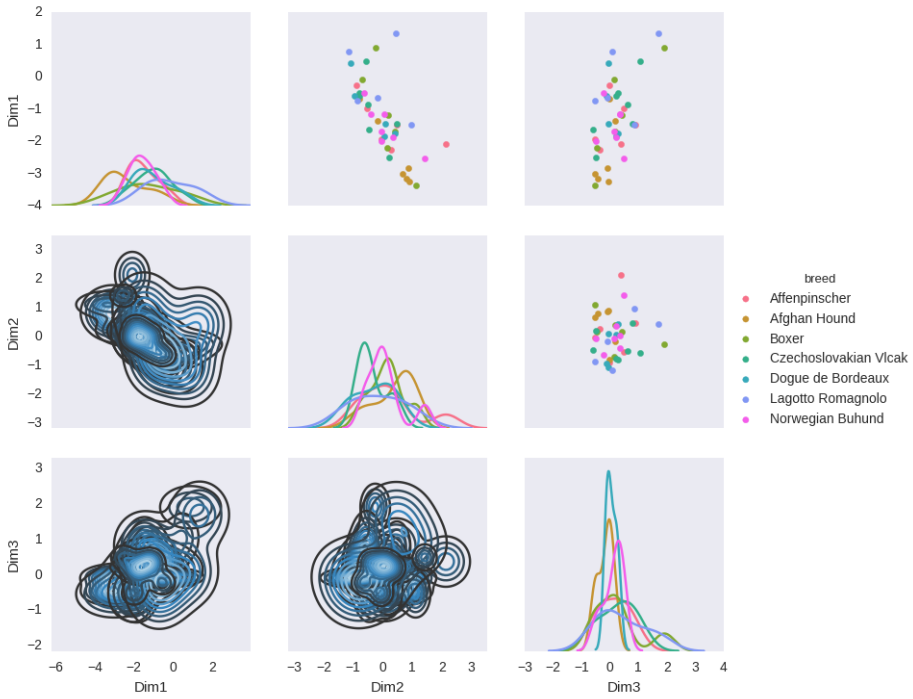

Figure 4: Pair plot of the foregrounds' first 3 Haralick PCA dimensions.

We can see from a subset of breeds that there is some clustering within breeds in the first 3 dimensions of the foreground Haralick texture PCA, but there is a lot of overlap. This does not bode well for classification based on these features.

Finally, a quick look at the grabCut() function, which separates foreground from background actually uses a k-means algorithm behind the scenes. We can see from comparing foreground and background RGB histograms that if there is some separability in color from foreground to background, grabCut() does much better than if the foreground and background color distributions are similar.

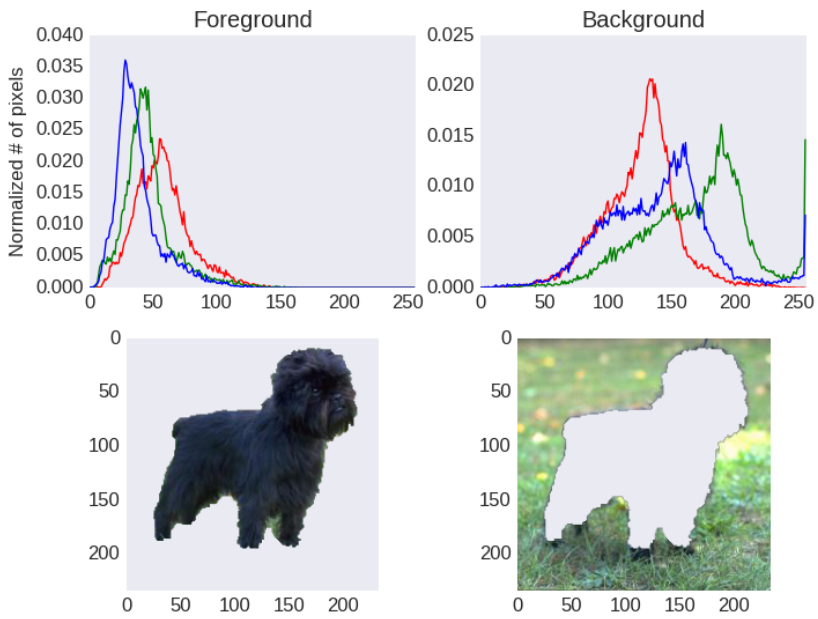

Figure 5: (Top) RGB color histograms of the foreground and background images after grabCut() has been applied. This is a dog from the 'Affenpinscher' breed.

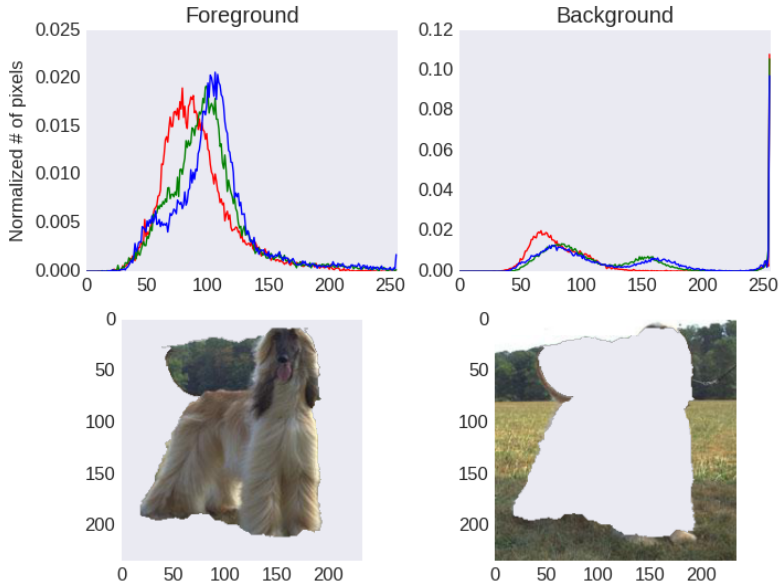

Figure 6: (Top) RGB color histograms of the foreground and background images after grabCut() has been applied. This is a dog from the 'Afghan Hound' breed.

Algorithms and Techniques

At this point, we've provided training data by scraping dog breeds and images, drawing rectangles around dogs in images, using the OpenCV grabCut() function to separate the

dog from the background, and extracting Haralick features and color histograms from the dog images (possibly transforming features via PCA). Machine learning algorithms (SVM, kNN, and RandomForest) are trained on the training data, using GridSearch to optimize the C and gamma parameters, using three folds for cross-validation. Typically, at least six images from each breed will be used for testing the classifier. For un-segmented test images, we will scale the image to a standard size if it is too large, then calculate Haralick features of center 60% of the image. If the algorithm was trained on PCA-reduced data, the features of the test images also must be transformed. Once we have the features ready, we can easily use the 'predict' method of the machine learning algorithm to classify the breed of the dog in the image.

One of the defining features of a dog breed is how it's coat looks. Part of this is color, and part of it is texture. Here, we are using the texture as an indicator of breed. An SVM for our classifier is appropriate here because we are doing supervised learning with multiple classes. Using the RBF kernel allows us to split the data non-linearly, which should be helpful here, due to the overlap in the feature data.

kNN also could be appropriate, because we would expect the training data to cluster around centers.

Finally, random forests are also appropriate, because the data can be separated based on many splits. Due to the large number of classes, we'd expect some overlap in many of the features. Some classes may have many overlapping features, with only a few small distinguishing traits. Random forests are a great choice for this scenario, because they can split these classes with mostly overlapping features near the bottom of decision trees.

Benchmark

The benchmark will be random guessing. For a single image, we have a 1/252 chance of getting the breed correct. For 1/3 of the 1447 images from training (many were thrown out due to grabCut or other failures) used for testing (482 images), we should get only 482/252 = 2 images correctly identified with random guessing, on average. This can be verified with the 'random_guess_benchmark.py' script.

Methodology

Data Preprocessing

I first had to collect a list of breed names and pictures. I started by using scrapy to scrape akc.org for breed names and the single picture they have for that breed. From the 'scrapy/scrape_dogs' folder, running the command:

```
scrapy crawl akc -o output.json
```

crawls the akc.org site and scrapes the necessary data. The ‘mainImPath’ parameter in ‘config.json’ will have to be changed for someone else to run the scrapy crawl.

Next, I used the breed names from that scrape to grab images from a Google image search. Running

```
nodejs google-images
```

will scrape Google images for dogs of each breed. You will need to change the credentials to your own Custom Search Engine credentials in the ‘credentials.cred’ file.

Starting from the raw images, training data is provided by manually going through images and drawing rectangles around dogs in the images. This was done using OpenCV, with the script ‘get_bounding_boxes_of_dogs.py’. Some of the scraped images from Google returned a webpage, or are covered in text, or have too many dogs in them. When loading the image using OpenCV (cv2 in Python), if the variable that should hold the image array is ‘None’, we know the ‘image’ was actually an html file or something else, and we skip the image. The bounding boxes for the dog in each image, along with the image file name, are stored in a pandas DataFrame and pickled to disk. I attempted to get at least 6 images for each breed, although some didn’t have enough good-quality images, and only 4 or 5 training images were obtained. The first time through the images, I found I only grabbed a few images for some breeds, so I went back through to even out the numbers of images per breed using the ‘get_bounding_boxes_of_dogs-even-out-population.py’ script, which prints to stdout the row in the pandas DataFrame for each image. This allowed me to check if the image already had bounding boxes drawn.

I allowed for drawing multiple bounding boxes on images, and with the touchpad on my laptop, I sometimes accidentally created small boxes. I cleaned the pandas DataFrame bounding boxes using the script ‘clean_bbs.py’, which removed any bounding boxes which had a total difference abs(x1-x2) + abs(y1-y2) of less than 20 pixels. It only found one bounding box like that, and removed it.

Next, I used the OpenCV function grabCut() to separate the dog from the background of the image (‘grabcut_ims.py’ file), for better quantification of Haralick textures of the dog fur itself. The grabCut() algorithm creates an image mask, which is an array that has the same height and width as the image. The area outside of the bounding box of the dog is set to a value of 0 (known background), the area inside is set as 3 (probably foreground), and the grabCut() function is run using the bounding box as an initializer:

```
cv2.grabCut(image, mask, rect, bgdmodel, fgdmodel, 5, cv2.GC_INIT_WITH_RECT)
```

The algorithm refines itself 5 times (thus the argument ‘5’), and returns the new mask. Next, it would try to refine 5 more times based on the mask resulting from the first run of

the function. Sometimes, it would be unsuccessful, and would skip that step. Finally, the image would be saved as RGBA images, with the alpha-channel as the mask for the foreground or background. This was done by splitting the image into color channels, and making the background or foreground transparent with the alpha channel:

```
b_channel, g_channel, r_channel = cv2.split(orig)
# if probably background (2) or background (0), set to transparent (0)
# otherwise make opaque (255)
a_channel = np.where((mask==2)|(mask==0), 0, 255).astype('uint8')
foreground = cv2.merge((b_channel, g_channel, r_channel, a_channel))
```

The file must then be saved as a '.png' filetype. This enables visual inspection of the background and foreground images, to see how well grabCut() is actually doing.

Next, we extract Haralick features from the foreground of the images. We load the foreground RGBA images from disk, and if an image is larger than 450 pixels in width, the image is resized so the width is 450 pixels. The squares that fit within the background and foreground masks are then extracted. If the grabCut algorithm failed to provide a region of the dog, and no squares fit in the foreground mask (because the foreground mask is too small), we skip that image. This resulted in a loss of 131 images, or about 8% of the initial images. Only extracting the Haralick features and color histograms from the foreground resulted in less loss: 89 images, or about 5.5%.

Drawing squares in the foreground was done via the following: squares are drawn over the entire image, and squares that lie within the foreground are added to a list of foreground squares. Haralick features are then calculated for each square. Each Haralick dimension is returned as a vector with shape 4. The Haralick features from each square in the foreground are then averaged to get a single 13x4 Haralick set of features for the foreground. These are later averaged across the 4 directions from each of the 13 dimensions, and transformed by taking the absolute value and logarithm of each value. The Haralick textures are then fit and transformed using PCA, and the first 3 dimensions of the Haralick texture PCA transform are used as features, since the first 3 PCA components make up 95% of the variation in the Haralick features. The interquartile range is calculated for each of the top 3 PCA dimensions, and entries with outliers in more than one dimension are thrown out. This resulted in a loss of 12 images. At this point, the total number of images is 1447.

Implementation

Extracting the Haralick features from many small regions and averaging them takes a relatively long time, sometimes taking 10s of seconds per image on my laptop. Running the Haralick feature extraction on the dataset linearly takes more than a few hours. I used the multiprocessing package to spread the feature extraction for each breed onto different threads to speed it up.

Once the features have been extracted and a PCA fit, we can train and test a model. The training data is a set of 1447 entries with length 3 vectors of Haralick features transformed via the PCA fit. Training and testing data for evaluation were created using train_test_split() from scikit-learn, with 67% of the data in training and 33% as test, with stratification across classes to get more balanced training and testing sets.

I then did a gridSearch on the SVC parameters C, gamma, and kernel. The gridSearch was completed using all available training data. The model was then refit using the best parameters with the training data and labels, and evaluated on the 33% hold out test dataset. I built a custom function that would count the number of correct predictions. It should also be possible to arrive at the same conclusion by using the 'score' method of the classifier, and multiplying by the size of the test set. I then did a similar process for kNN and RandomForest with grid searching.

Refinement

When only grid searching the C parameter with the SCV, it found the lowest C to work best (0.05, or 0.1 if 0.05 was not included). The model only correctly predicted 3 breeds, doing barely better than random guessing. I did another gridSearch with the kernel, C, and gamma parameters as such:

```
params = {
            'kernel': ['rbf', 'sigmoid'],
            'C': [0.05, 0.1, 0.2, 0.5, 1.0, 2.0],
            'gamma': ['auto', 1, 1/2.]
            }
```

This resulted in optimal parameters of {'kernel': 'rbf', 'C': 0.5, 'gamma': 1} (gamma=auto). It improved the correct predictions from 3 to 4 out of the 486 test data entries.

Next I tried the kNN algorithm and searching n_neighbors from 3 to 10, and weights: ['distance', 'uniform'], finding 3 neighbors and weight=uniform as the best. This only correctly predicted 2 breeds, equal to random guessing.

Finally, I tried the RandomForest, grid searching the parameter space 'max_depth': [20, 30, 40, 50, 60], 'n_estimators': [30, 40]. It found the best parameters to be max_depth=50 and n_estimators=30, and only correctly predicted 2 breeds.

My first thought was that the feature dimensions were too small, so I tried using the 13-dimension Haralick feature instead of the 3 components of the PCA. I did not remove outliers this time, so the full training set was 1459 entries, and the test data set was 494 entries. I used the rbf kernel, and the gamma range had to be increased for the higher number of dimensions (['auto', 1, 1/5., 1/10.]), and gridSearch resulted in different

optimized parameters: {'C': 1, 'gamma': 1}. The correct predictions improved to 13, or 2.6%. Since the gridSearch resulted in a C value of the lowest amount, I changed the parameter place for C to [0.1, 0.5, 1.0], but it gave the same optimized parameters.

Retrying the kNN, I found parameters weight='distance' and n_neighbors=6 to be best, which predicted 21 breeds correctly, or 4.3%. For RandomForest, I found the best parameters to be max_depth=40, n_estimators=40, which correctly predicted 18 breeds (3.6%).

I then thought that if more Haralick dimensions is good, increasing it still further might improve the classification. I went back and re-did the Haralick analysis, leaving the full dimensionality of the Haralick texture in place (13x4 feature), and only extracting features from the foreground, using the file. I didn't extract color histograms during the processing either. This resulted in 1501 entries in the test dataset (something about the background and/or color histogram extraction must've been preventing some of the entries from making it through the Haralick extraction before), and 503 test entries. The feature length is now 52.

A grid search of the SVM with C: [0.1, 1, 10.0] and gamma: ['auto', 1, 1/10., 1/25.] resulted in {C': 1, 'gamma': 'auto'} as the best parameters. The correct predictions decreased to 6 (1.2%).

For the kNN, the best parameters were {'n_neighbors': 8, 'weights': 'distance'}, and it correctly predicted 9 breeds (1.8%). The random forest parameters came out to be {'n_estimators': 30, 'max_depth': 50}, and it correctly predicted 23 (4.6%). A table summarizing the results shows the Random Forest does well with increasing dimensionality:

feature dimensions	best SVM score	best kNN score	best RandomForest score
3	0.80%	0.40%	0.40%
13	2.60%	4.30%	3.60%
52	1.20%	1.80%	4.60%

It seems the SVM and kNN do best with intermediate feature dimensions, and RandomForest just keeps getting better with higher dimensions.

I also knew that adding in information about the color of the dog would improve the classifier, and had already built in functionality to measure the color histograms of the foreground in the images in the script 'extract_fg_hists-linear.py'. I went back and did PCA on the color histograms, using the fist 20 dimensions of the PCA, and incorporated that in the training data, with the 52-dimension Haralick features. I found the first 20 dimensions of the PCA transform of the foreground images made up about 91% of the variation.

Trying the SVM, kNN, and RandomForest again with optimal parameters from GridSearch resulted in scores of 8 (1.6%), 9 (1.8%), and 28 (5.6%). I empirically found increasing the max_depth to 50 increased the score to 38 (7.6%).

Results

Model Evaluation and Validation

The final model was chosen to be the Random Forest of Decision Trees using a 20-dimension PCA of the RGB color histograms and the 52-dimension Haralick texture, because it was the most accurate. Although the tree depth is large (max_depth=30), there are many classes (252) and features (72), so a deep tree is reasonable here. Looking at the correlation between total number of pictures and number of correct predictions, we can see that a higher number of images in the test set seems to correlate to a higher amount of correct predictions:

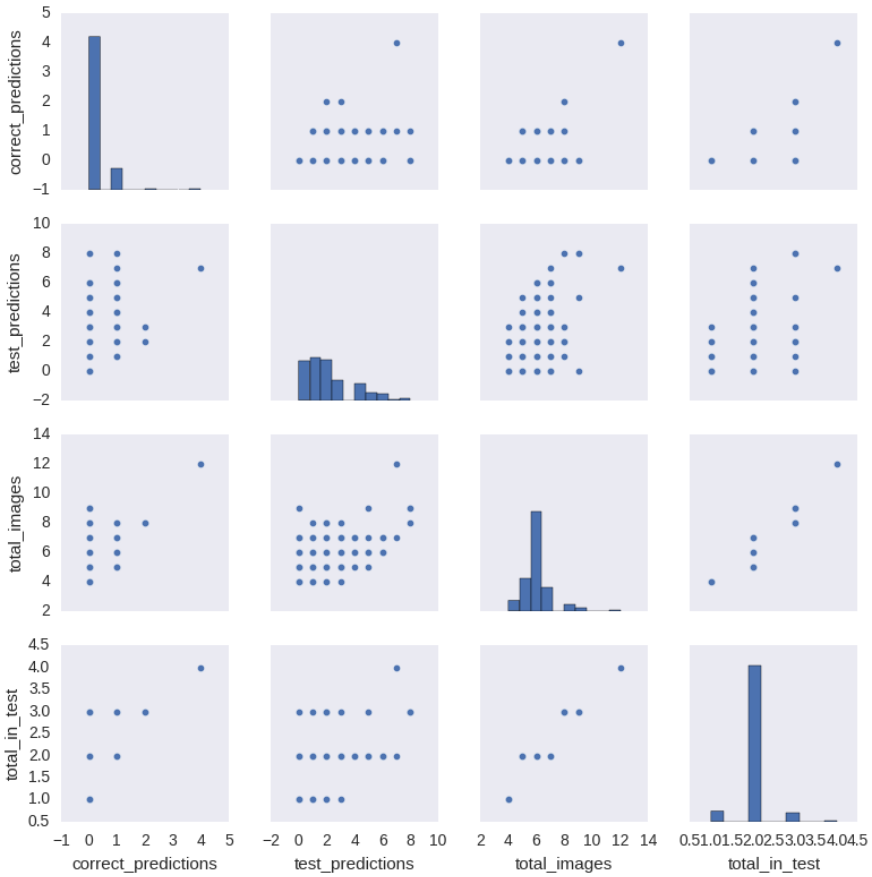

Figure 7: Pair plot of number of images in test and total, and correct predictions in the test set for all breeds.

The outlier with 1 2 images is the 'Whippet' breed, which also was predicted correctly 4/4 times in the test set. We can see from the test_predictions histogram that most

breeds are either not predicted at all during the testing, or only predicted once, although almost all the breeds have 2 images in the test set.

To test how well it generalizes to unseen data, I grabbed 3 random images from each breed and evaluated the performance, using the script ‘check_robust.py’. I did not perform dog outlining with rectangles or grabCut(). The classifier was trained on the full training dataset. In total, 753 images were grabbed (one breed must’ve not had enough suitable images, because only 251 breeds were represented). The classifier only correctly predicted 3 breeds, which equal to random guessing, which would yield about 3 correct breeds on average.

To test the robustness of the model, I introduced 10%, 20%, 30%, and 40% relative Gaussian noise into the training data features, and retrained the classifier. The correct predictions on the test data decreased from 38 (no noise) to 19, 21, 27, and 19 for the respective noise levels from 10-40%. It doesn’t seem very robust, since 10% noise decreases the accuracy substantially.

I also introduced Gaussian noise into the testing features, and scored the classifier trained on clean training data. With 10%, 20%, 30%, and 40% relative Gaussian noise on the testing data, the correct predictions were 18, 14, 10, and 9. The sensitivity to noise in the testing data set was much higher.

In general, results from the model cannot be trusted due to the low accuracy.

Justification

On images where the dog has been segmented from the background, the classifier beats the benchmark of random guessing. For images that don’t have this segmentation, the classifier is no better than random guessing. It solved the problem of beating the benchmark, but in a weak way. The classifier unfortunately is not good enough to generalize to unseen data that hasn’t been specially pre-treated.

Conclusion

Free-Form Visualization

I was curious about the uniformity of Haralick texture across the dogs’ bodies. I created a function that maps the relative variation of the first PCA component of the Haralick texture to each rectangle as a brightness of the color green. We can see from the examples that the texture is very different around parts of the face, as we might expect. It is also obvious that lighting conditions cause great variation in the Haralick texture. This is probably why the current implementation of the Haralick texture is not working great as a descriptor of the data.

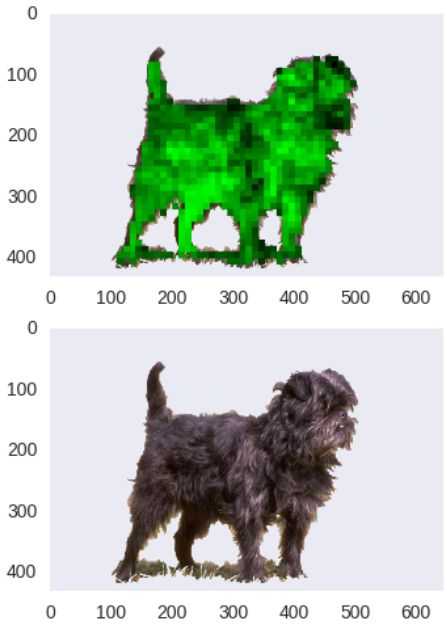

Figure 8: Map of the variation of the first PCA component of Haralick texture on an image of an Affenpinscher.

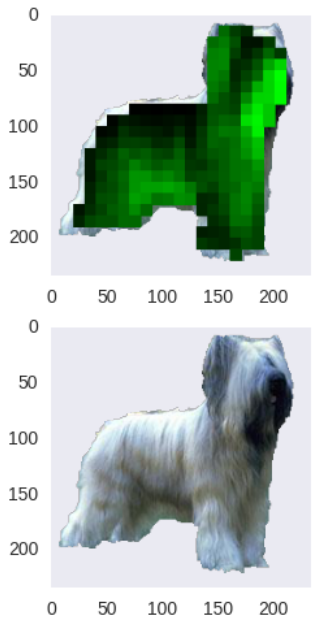

Figure 9: Map of the variation of the first component of the PCA of Haralick texture on an image of a Briard.

Reflection

From the beginning of the project, I learned a lot. Generating my own dataset was difficult, and I didn't know how to use scrapy before starting this. I learned how to download information and images from websites using scrapy to provide the backbone of data for the project. I also learned how to scrape Google images using Node.js. Both of the scraping tasks were quite difficult and time consuming to figure out. Next, I learned how to use some interactive cv2 features and grabCut, as well as a little bit about how grabCut works (kMeans clustering).

Experimenting with keypoint detection was interesting, since that is a conventional way to classify images. Unfortunately, the amount of keypoints seemed way too large to be able to use it for a classifier without rigorous grabCut preprocessing—often times, most of the keypoints would actually be detected in the background.

I realized after doing the PCA and outlier analysis that it would probably make more sense to do the outlier analysis *before* fitting the PCA, so that extreme variation by a few data points doesn't skew the PCA fit.

I improved my data wrangling skills—during the initial phase of Haralick texture extraction, I had forgotten to include the breed information with the data. To reconstruct the data with the breed information attached to the Haralick features, I had to learn a lot

of data massaging with pandas. This, along with other data massage I had to do for seaborn plotting and feature preparation for the classifiers really improved my skills with the pandas package, and data wrangling in general. The final model met the goal of beating random guessing, but it's not usable in the real world. I really didn't expect it to be that bad (7% accuracy); I thought it would have at least 25% accuracy.

Improvement

The color properties in the features could have spatial encodings embedded in them. For example, breaking up the image into subsections, and extracting the color histogram principal components from those areas, or just average RGB (or other color space) values may help discriminate dogs with spots from those without.

Another improvement could be using a bag-of-words type model with the texture features, as well as color. The training images of dogs could be broken up into subsections, and features extracted on each subsection. Then, kMeans classification would be performed with the subsection features. For an un-classified image, the image would be broken up into subsections, and features extracted from each subsection. Then, the kMeans classifier would find the closest breed for each subsection, ordering or weighting the subsections by their Euclidean distance to the nearest breed in the kMeans clusters.

However, I think with complex data and a plethora of classes, a neural network may be best suited for the problem.

I know there is a much better solution out there than this one. Even though the Microsoft side project app what-dog.net doesn't provide scoring metrics, it's not hard to tell they have better accuracy than this implementation. In the future, I would like to utilize a neural network for this problem, and utilize keypoint detection with a bag-of-words-based model.

References:

[1] http://www.statista.com/statistics/198100/dogs-in-the-united-states-since-2000/
[2] https://www.psychologytoday.com/blog/canine-corner/201209/how-many-dogs-are-there-in-the-world
[3] http://www.akc.org/
[4] https://www.princeton.edu/genomics/kruglyak/publication/PDF/2004_Parker_Genetic.pdf
[5] https://www.what-dog.net/
[6] https://en.wikipedia.org/wiki/List_of_dog_breeds
[7] https://en.wikipedia.org/wiki/List_of_dog_breeds_recognized_by_the_FCI
[8] https://en.wikipedia.org/wiki/List_of_dog_crossbreeds
[9] http://www.dogbreedinfo.com/abc.htm
[10] https://dogs.thefuntimesguide.com/2006/09/hybrid_mixed_dog_breeds.php
[11] http://www.thekennelclub.org.uk/services/public/breed/Default.aspx
[12] http://www.kennelclubofindia.org/index.php?option=com_content&view=article&id=88&Itemid=95