Image Classification on 20 Categories Pictures

Chun-Ning Tsao, Xiaoer Hu

Abstract

An image classifier with high accuracy (43%) is built in this work through data processing, feature extraction, and model evaluation. Random forest classifier turns to have the best performance among all the non-NN classifiers, and the CNN model achieves even 53% accuracy.

Keywords: Image Classification, EDA, feature extraction, classifier training, random forest

1. Introduction

Image classification refers to classifying an image by the object category that it contains based on finite training data and is of growing interest recently due to the rising popularity of camera devices and video databases. It becomes increasingly challenging as the number of object categories increases, therefore more efficient and accurate image classification algorithms are required [1]. In this work, we performed data pre-processing, visualization, exploratory data analysis, feature generation and selection, image prediction, and multiple classification models assessment to build a highly accurate image classifier using 1501 images as the training set which contains 20 different categories. The category distribution of the training set is shown in Fig. 1. The accuracy of our model is 43% on the validation data set. With the neural network implemented, the accuracy becomes 53% on the same validation set.

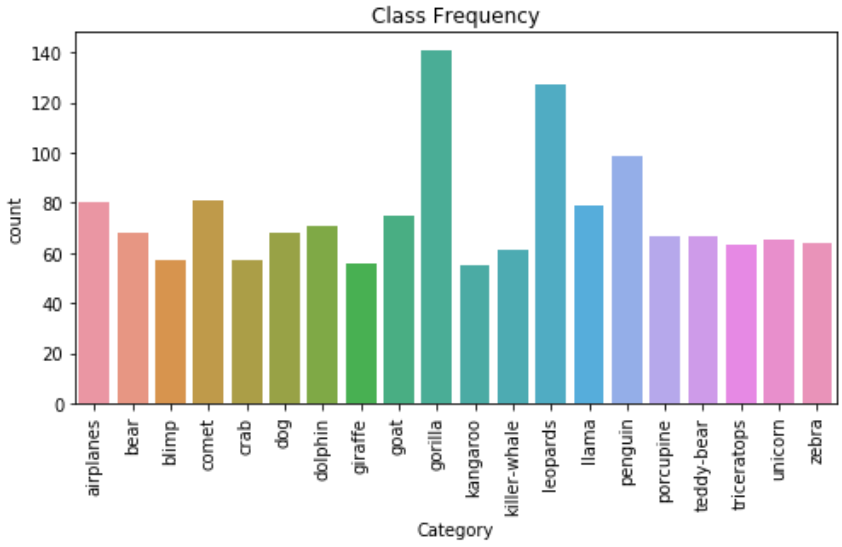


Fig.1. The class frequency for different categories in the training data set.

2. Methods

The training data set we start with contains 1501 images with 20 different categories: airplanes, bear, blimp, comet, crabs, dogs, dolphins, giraffes, goat, gorilla, kangaroo, killer-whale, leopards, llama, penguins, porcupine, teddy-bear, triceratops, unicorn, and zebra. The test data we need to perform our best classifier onto has 716 images. Most images are represented as a 3-dimensional array, where the first and second dimensions refer to the number of pixels in the row and column of the image, and the third dimension describes the red, green, and blue (RGB) color intensity of the images, between 0 and 255. However, there are 16 images in the training set and 11 images in the test set that are gray. We convert the gray images into RGB images when loading the files to match the data format of most images, then we can suppose all images are in RGB format when generating the features. Besides, some of the pictures have white or black borders, which are not desirable since they may misguide the color-related features. We detected the borders by checking whether the upper left or bottom right of the image is white or black, and cropped them by setting a color threshold. This step is also crucial since the image shape will be important to generate features such as size, aspect ratio, width, height, etc. for our classifier. Notice here we also need to set a threshold value of the cropped image size to prevent over-cropping. For example, comet_26 is mostly black and would be over-cropped if we used the same color threshold as other images. For simplicity, this type of image will be just saved with its original values. The images from the training dataset and testing dataset are then stored into two Pandas dataframes after the above two pre-processing steps. Before saving the test data into the dataframe, we also sorted the test sets based on its original file name. To better transfer data between notebooks for this project, the dataframes were saved into pickled objects to save the storage space, and due to the memory management issue of the datahub, we split the train dataframe into 3 pickle files, and test dataframe into 2 pickle files.

We generated several features based on the images, including 13 scalars and 3 matrices, as summarized in Table 1 below.

Table 1. The summary of generated features

Feature Name	Type	Description
Size	Scalar (>0)	Image size (width * height)
AspectRatio	Scalar (>0)	Aspect ratio (width / height)
Height	Scalar (>0)	Image height
Width	Scalar (>0)	Image width
AvgRed	Scalar ([0,255])	Average of the red-channel
AvgGreen	Scalar ([0,255])	Average of the green-channel

AvgBlue	Scalar ([0,255])	Average of the blue-channel
HistRGB	Matrix (shape = 64*1)	RGB histogram with 4 bins for each channel
AvgHue	Scalar ([0,179])	Average of the hue-channel
AvgSaturation	Scalar ([0,255])	Average of the saturation-channel
AvgValue	Scalar ([0,255])	Average of the value-channel
VarHue	Scalar (>=0)	Variance of the hue-channel
HistHSV	Matrix (shape = 72*1)	HSV histogram with 8 bins for H, 3 bins for S and 3 bins for V channel
Sharpness	Scalar (>0)	Sharpness based on Laplacian filter
Contrast	Scalar ([0,1])	Contrast according to difference of block-based lightness
Corners	Matrix (shape = 98 or 28224*1)	Sorted coordinates of 50 best corners

As Fig. 2 boxplot shows, some categories, for example, airplanes, kangaroo, leopards, and llama have very similar image size for all its images, while the other categories have very different image size distributions, for example, for the images that contain bears, some of them have image size at around 4×10^4, while some have 2×10^7 pixel size. Therefore, the size feature is useful for some categories. Another important feature is the aspect ratio, whose boxplot by category is shown in Fig. 3 below. Aspect ratio is useful because for objects like airplanes, crabs, leopards, and zebras, they tend to have long shapes in the horizontal direction, resulting in large aspect ratios. We also listed the width and height of images as features for classifications, and their distributions by category are shown in Fig. 4 and 5. Similarly, the distribution of airplanes, kangaroos, leopards, and llamas has less variance than other categories. The average values of red, green, blue, and the histogram of the RGB are all selected as features and are shown in Fig. 6 and 8 since they can describe the color information contained in the images. Similarly, Fig. 7, 9, and 10 show the average values of hue, saturation, value (brightness), and the histogram of the HSV, as well as the variance of hue-channel. HSV analysis is important because the RGB along in an image is correlated with the light hitting the object, so descriptions in terms of hue/saturation/value are more relevant, and hue is especially important in the case where the illumination level varies from point-to-point or image-to-image [2]. Sharpness might also be a useful feature as it can show some texture information of the object in the image [3], which is shown in Fig. 11. Contrast distribution by category, as Fig. 12 shows, is also different for different categories, but if we consider the 5% to 95% data, the contrast varies between 0.2 and 1.0 for most categories. We also extracted the 50 most important corners for each image, and one of the examples is shown in Fig. 13. Corners should be a critical feature since it detects the shape of the object, which should be the most important clue to classify the category in intuition [4], but we could not find an effective way to utilize corner feature. One corner feature implementation method we tried is to find the corners in the image and sort by the coordinate locations from the upper left to the lower right. If we flatten the list of coordinates into a 1-D feature vector, each dimension over different images have

different images and are not comparable. Another implementation is to create a binary matrix showing whether each point is a corner.

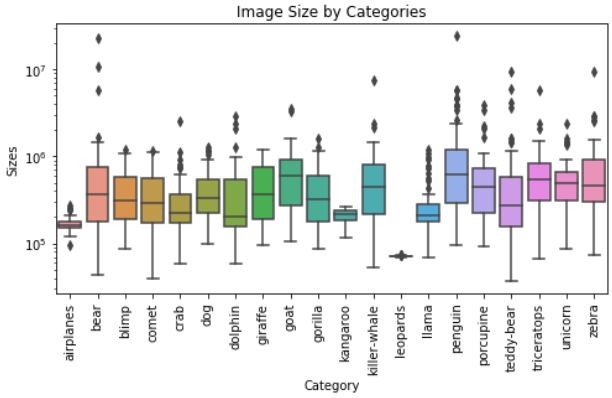


Fig.2. The image size by categories.

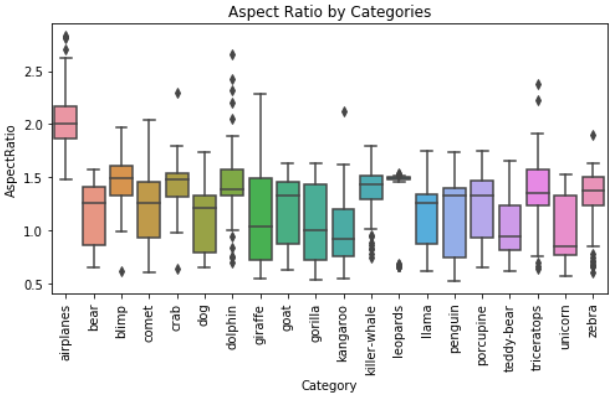


Fig.3. The image aspect ratio by categories.

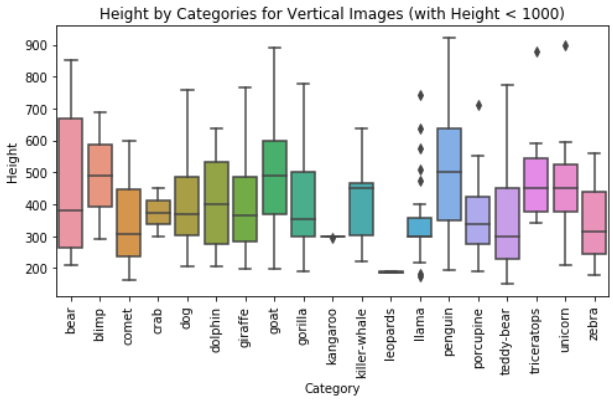


Fig.4. The image height by categories.

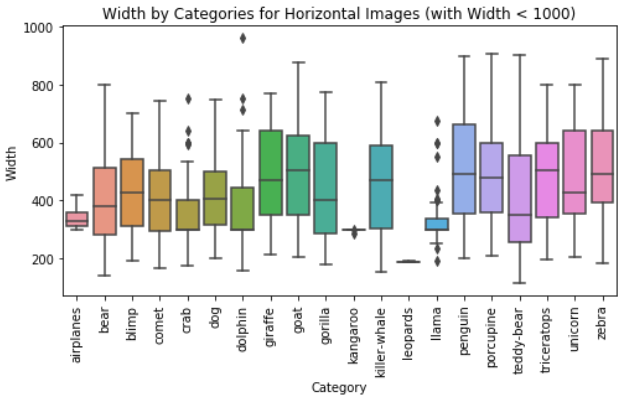


Fig.5. The image width by categories.

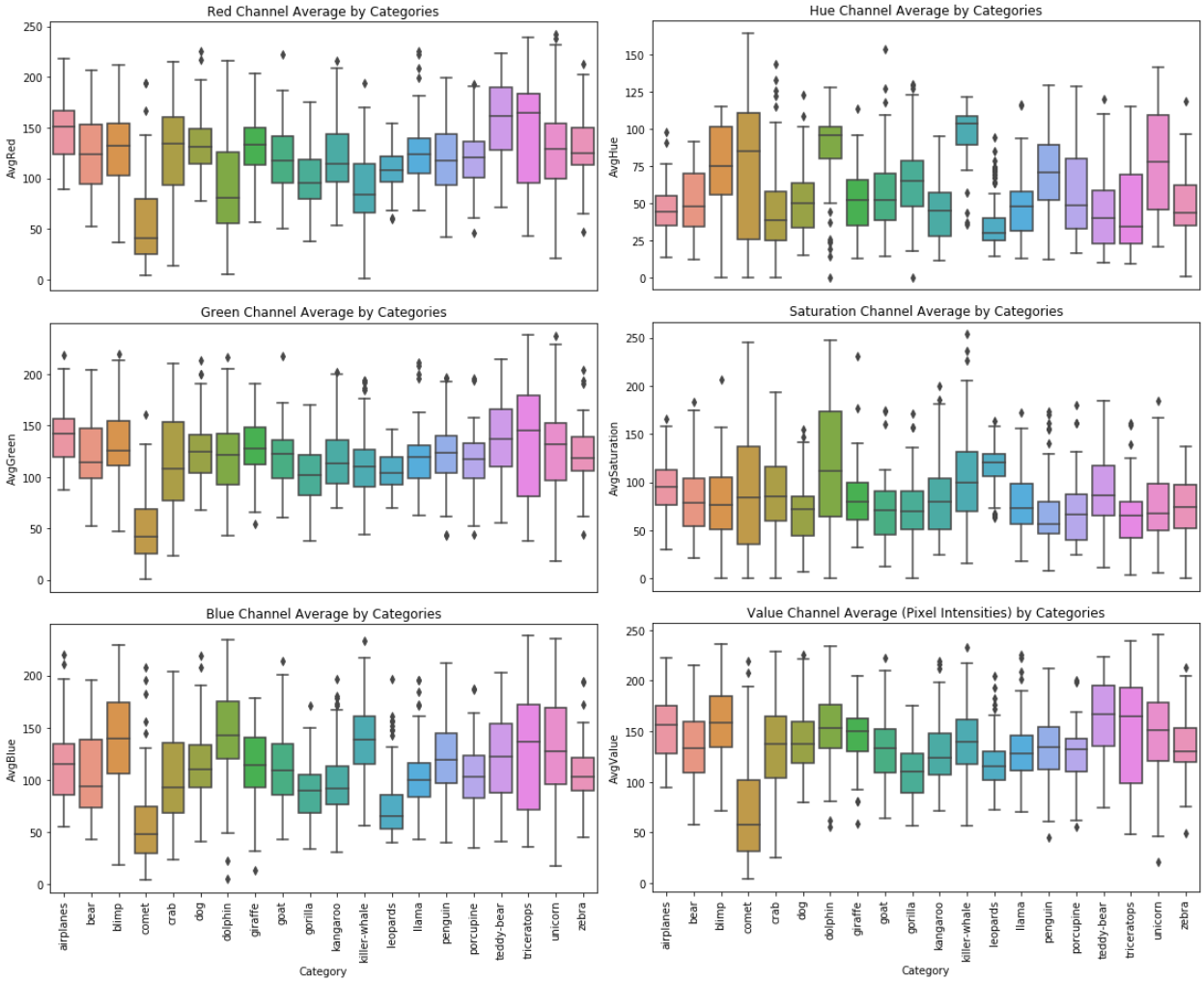


Fig.6. The image average RGB by categories.

Fig.7. The image average HSV by categories.

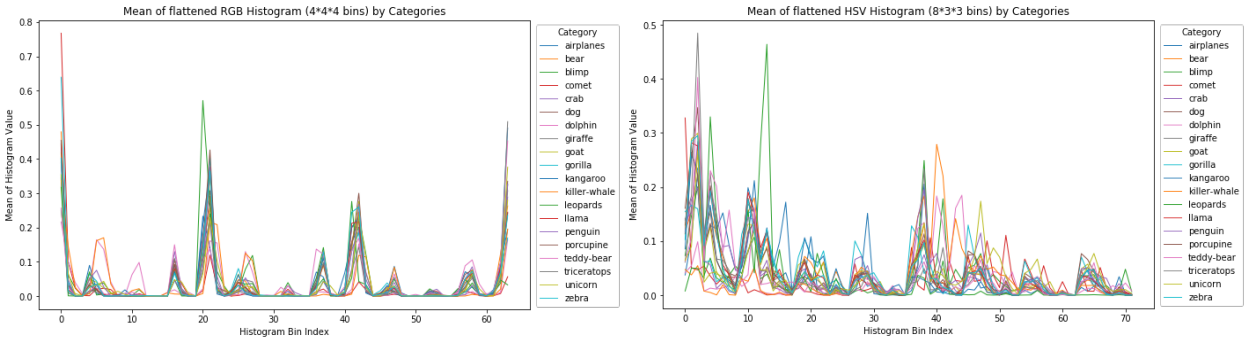


Fig.8. The image RGB histogram by categories.

Fig.9. The image HSV histogram by categories.

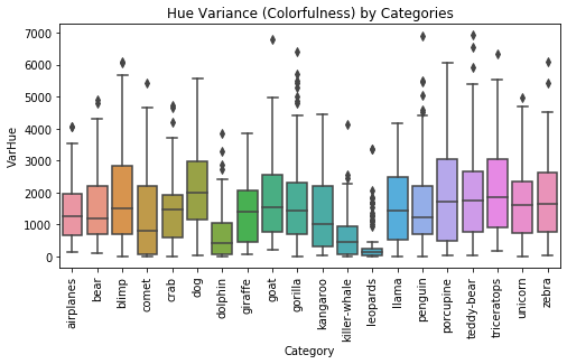


Fig.10. The image hue variance by categories.

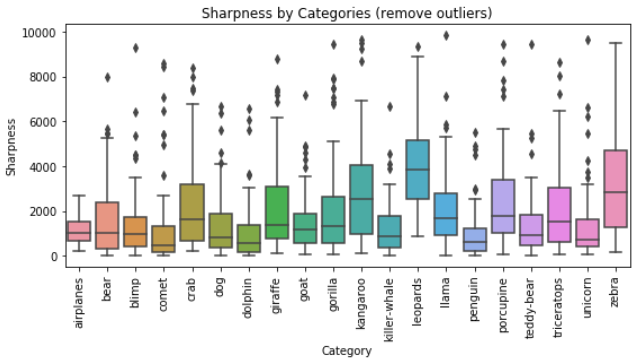


Fig.11. The image sharpness by categories.

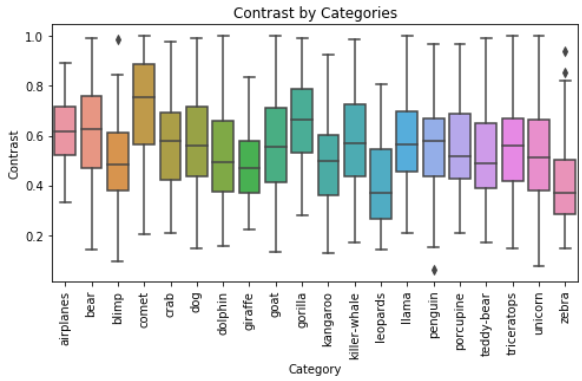


Fig.12. The image contrast by categories.

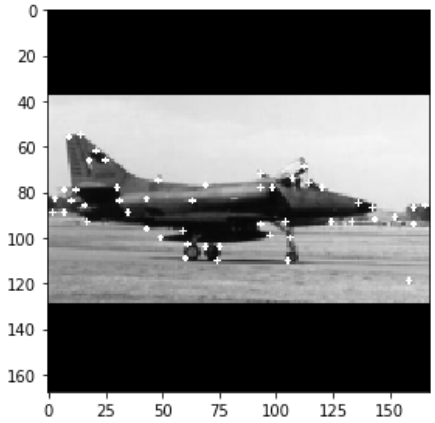


Fig.13. Example of 50 best corner detection.

For all the HSV related features, we transformed the original images into the HSV format first, then applied the corresponding functions to the HSV images to extract the corresponding features. For sharpness, contrast, and corner features, we converted the image into grayscale before applying the functions. A resize function is implemented for corner feature extraction, to produce unified-shaped images (the unified images all have 168*168 pixels).

3. Results and Discussion

To better utilize our features, all numerical features are normalized by the StandardScaler() function. For the matrix features (which are in the form of 1-D vector), we need to treat each element as a new column. For the corner feature, we also normalized the scalars after the column transformation. The classifiers we used include logistic regression, k-nearest neighbors (K-NN), decision tree, random forest, and support vector machine (SVM).

First, we put all numerical variables into feature evaluation functions to get the most important features automatically. The most important features that sklearn found are Size, AspectRatio, Height, Width, AvgHue, and VarHue. The aspect ratio of the image is important because it tends to be large for wide objects such as airplanes and leopards. The average and variance of the hue-channel are important because the hue value describes the color of the image, which varies between

categories. Interestingly, the Size, Height, and Width are important according to sklearn. This phenomenon does not match our common sense, because the dimension of the image usually depends on the digital cameras, it should not be closely related to what is actually on the image. If we exclude these three features and run the feature selection function again, the most important features become AspectRatio, AvgHue, and VarHue. Therefore, we tried three different scalar feature selection methods: all scalar features; all scalar features without Size, Height and Width; the 3 most important features when excluding Size, Height and Width (i.e. AspectRatio, AvgHue, and VarHue). The summary of the above accuracy results of 5-fold cross-validation on the training set is listed in Table 2. It turns out the accuracy by using (a). all scalars and (b). all scalars except Size, Height, and Width are much higher than (c). only using three features.

We also tried to utilize the matrix features in the classification process. First, the effectiveness of each matrix feature was checked respectively, also listed in Table 2. Surprisingly, the corner feature looks not as effective as expected, while the histogram of RGB and HSV are good features. Therefore, we add these two matrices together with the above two only scalar features cases: (a). all scalars and (b). all scalars except Size, Height, and Width. The highest accuracy occurs when we used the histogram of RGB and HSV, as well as all the above-listed scalar features, which is around 40%. We also tried to add square terms to the most important 6 features and add them with the above feature combinations, as shown in the last 2 rows of Table 2. The accuracy with the added square terms stays almost the same with the original feature combination. Therefore, to keep model simplicity, we will keep using the histogram of RGB and HSV, with all the listed first-order scalar features to predict our final test set.

Table 2. The summary of 5-fold cross-validation accuracy on learning set

Selected Features	Logistic regression	K-NN	Decision tree	Random forest	SVM
all scalars	0.36	0.28	0.32	**0.38**	0.37
scalar w/o Size/Height/Width	**0.33**	0.24	0.25	**0.33**	0.32
AspectRatio+AvgHue+VarHue	0.23	0.23	0.24	**0.26**	**0.26**
HistRGB	0.29	0.23	0.23	**0.34**	0.30
HistHSV	0.32	0.26	0.24	**0.34**	0.32
Corners	0.14	0.13	0.12	0.17	**0.19**
HistRGB+HistHSV	**0.33**	0.27	0.25	**0.33**	**0.33**
HistRGB+HistHSV+scalar w/o Size/Height/Width	0.37	0.28	0.27	**0.38**	0.34
HistRGB+HSV+all scalars	0.39	0.29	0.30	**0.40**	0.37

HistRGB+HSV+all scalars+square of Size/Height/Width/ AspectRatio/AvgHue/VarHue	**0.40**	0.30	0.31	**0.40**	0.36
HistRGB+HSV+all scalars+square of AspectRatio/AvgHue/VarHue	0.40	0.31	0.30	**0.41**	0.37

The most interesting features we came across are the height of the image and the average of the hue-channel. We got the height feature by counting the number of pixel columns in the image. As we stated, the height of an image should not determine what objects are in the image, but it turns out height is the second most important feature from the feature selection function, and if we exclude the height in the classification functions, the classification accuracy would drop. The mean value of the hue-channel is interesting, and we got this feature by converting the original image into an HSV format and count the average hue values.

The corner feature, however, is surprisingly ineffective in our models. Usually corners are important features in image classification because they can provide outlines of the objects in the image. However, in our models, even we calculated 50 most important features for each image, tried two different methods to generate corner feature matrix, resized all the images for corner detection, and normalized the corner columns after transforming the matrix into columns, it is still not helpful in the classification process. As stated before, one possible reason is that we did not fully use the corner information extracted because the list of coordinates is hard to interpret.

Regarding the differences in the classifiers, we can also see their performance comparison in Table 2. We found that K-NN is usually the most time-consuming and worst performed classifier. One possible reason for this is the number of dimensions in our feature matrix is too big for K-NN, there would be 149 dimensions for our final selected features. The single decision tree usually has similar performance with K-NN and needed to put special care about setting proper max_depth and min_samples_split to since it's easy to over-fit the training set.

The random forest can be viewed as an improved version of decision tree and usually performs well with less tendency to over-fitting. Interestingly, logistic regression, SVM and random forest have similar performance with all the combination of features we tried since we usually expect SVM to have better performance on complex data. Also note that we tried various solvers for regression and various kernels for SVM but the performance remains similar. Combined with these observations and the confusion matrix, we think the reason behind the similar performance between logistic regression, random forest and SVM might be that there are certain categories especially easy to classify and others very hard to classify, so all classifiers can only deal with simple cases.

We think the major limitations of our methods are two phrases. First, the training set is too small and if we split 10% as the validation set, there would only 150 validation images, which means only 7 images for each category on average. Under this situation, the performance difference can

go to 5% when we modify the size of the validation set, and the model also may not have good generalizability for categories with few samples. Second, we think the most critical feature for this task should be shape-related features such as Corners, SIFT, HoG since they describe the shape of the main object in the image, which is also the rationale behind human doing classification. However, we do not find an effective way to utilize this kind of feature and our model only relies on the size and color-related features, which greatly constrain our capability to distinguish categories that have similar color features, which are as we can see in the graph summaries above in Section 2.

A better way to construct the classifier might be using neural network directly trained on raw images rather than using hand-crafted features. We built a fully-connected DNN and a CNN, but we would only discuss the CNN below since it performs much better than DNN. The performance of DNN is put in the appendix. For the input, we first resize all images to form a unified input shape and we use an image generator to do data augmentation, including horizontal flip, rotation, zoom, and sheer, to improve the generalizability of our model. Our model consists of 4 CNN blocks, each block contains a Conv2D, BatchNormalization, LeakyReLu, MaxPooling2D, and a Dropout layer. Then there are 2 fully-connected blocks after flatten, and finally a Dense layer with 20 units and softmax. The simple CNN model early stops at 20 epochs and can already achieve 53% accuracy on the validation set. We believe the performance can still be better with a better network structure, parameter tuning, and utilizing pre-trained models.

To better understand our final classifier and CNN model, we plot the confusion matrix of their prediction over the same validation set (note they are trained on the same training set). As shown in Fig. 14 and Fig. 15, the non-NN classifier (we use random forest here) performs very well on certain categories such as airplanes, comet, gorillas, leopards, and triceratops, but cannot distinguish some other categories at all. We first thought the excellent accuracy on airplanes and leopards is due to their special distribution in Size, but the confusion matrix remains similar even if we remove Size feature. For the CNN model, it can also handle all categories that are well recognized by a non-NN classifier and can further distinguish some more categories such as killer-whale, unicorn, and zebra that non-NN classifier cannot handle. However, it also cannot distinguish some categories at all. For example, the accuracy of bear and goat remains zero. It might be too hard for the model to learn the characteristics of these categories from the current small training set.

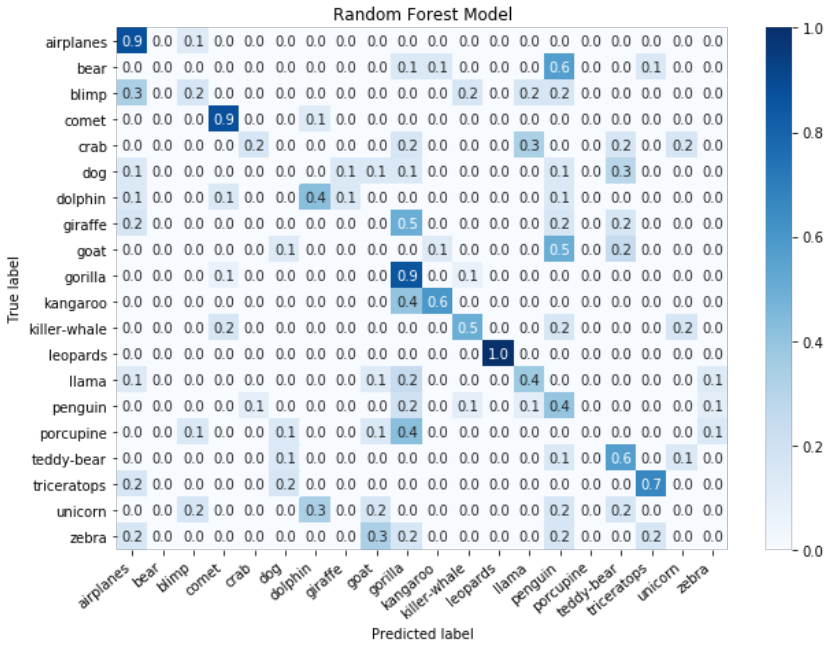


Fig.14. The confusion matrix generated by random forest model on validation data set.

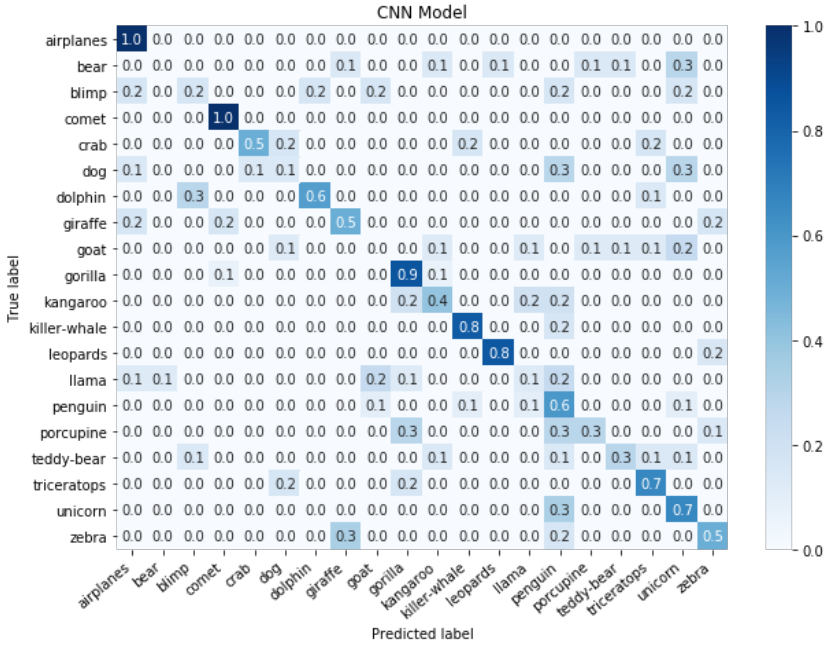


Fig.15. The confusion matrix generated by CNN model on validation data set.

4. Conclusion

We performed data cleaning, EDA, feature extraction and selection, as well as different regression models evaluations to build an image classifier with 43% accuracy in the validation data set. With the neural network implemented, the accuracy can be even 53%. During the feature selection process, we learned that the aspect ratio, height, width, size, average and variance of the hue-channel are the most important scalar features. The histograms of RGB and HSV are very important matrix features if we transform the matrices into columns and append the new columns with all scalar features we generated. The most surprising thing we realized in this project is that the corner matrix generated is very ineffective, although we implemented several methods to try to improve its performance. Therefore, future work can be done on better understanding how to properly use the corner features in image classifications. Among the five different classifiers we used, random forest seems to have the best performance for almost all the feature combinations we tested.

Reference:

[1]. Zhang, W., Xue, X., Sun, Z., Guo, Y., Chi, M., and Lu, H. (2007). "Effective Feature Extraction for Image Classification".

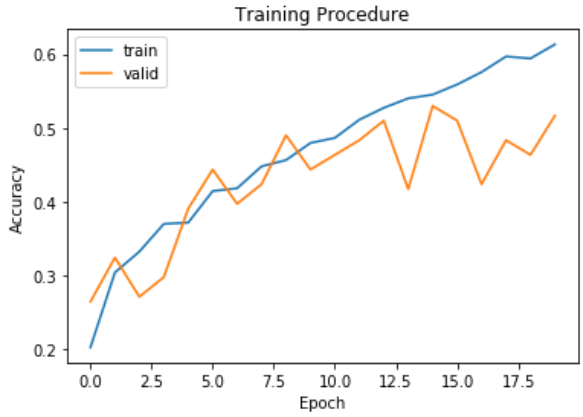
[2]. Cheng, H., Jiang, X., Sun, Y., and Wang, J. (2010). "Color Image Segmentation Advances & Prospects".

[3]. Rosebrock, A. (2018, August 2). "Blur detection with OpenCV". Retrieved from https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/.

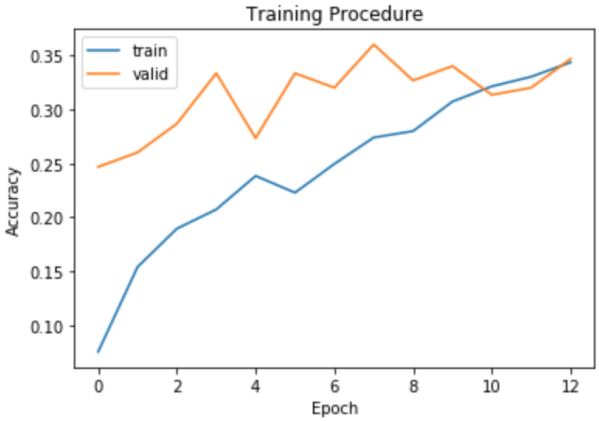
[4]. "Shi-Tomasi Corner Detector & Good Features to Track". (n.d.). Retrieved from https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi.

Appendix:

The Training Procedure for CNN Model:



The Training Procedure for DNN Model:



The CNN Model Structure:

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 168, 168, 64)	4864
batch_normalization_7 (Batch	(None, 168, 168, 64)	256
leaky_re_lu_7 (LeakyReLU)	(None, 168, 168, 64)	0
dropout_7 (Dropout)	(None, 168, 168, 64)	0
conv2d_6 (Conv2D)	(None, 168, 168, 64)	36928
batch_normalization_8 (Batch	(None, 168, 168, 64)	256
leaky_re_lu_8 (LeakyReLU)	(None, 168, 168, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 84, 84, 64)	0
dropout_8 (Dropout)	(None, 84, 84, 64)	0
conv2d_7 (Conv2D)	(None, 84, 84, 128)	73856
batch_normalization_9 (Batch	(None, 84, 84, 128)	512
leaky_re_lu_9 (LeakyReLU)	(None, 84, 84, 128)	0
max_pooling2d_5 (MaxPooling2	(None, 42, 42, 128)	0
dropout_9 (Dropout)	(None, 42, 42, 128)	0
conv2d_8 (Conv2D)	(None, 42, 42, 256)	295168
batch_normalization_10 (Batc	(None, 42, 42, 256)	1024
leaky_re_lu_10 (LeakyReLU)	(None, 42, 42, 256)	0
max_pooling2d_6 (MaxPooling2	(None, 21, 21, 256)	0
dropout_10 (Dropout)	(None, 21, 21, 256)	0
flatten_2 (Flatten)	(None, 112896)	0
dense_4 (Dense)	(None, 256)	28901632
batch_normalization_11 (Batc	(None, 256)	1024
leaky_re_lu_11 (LeakyReLU)	(None, 256)	0
dropout_11 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 128)	32896
batch_normalization_12 (Batc	(None, 128)	512
leaky_re_lu_12 (LeakyReLU)	(None, 128)	0
dropout_12 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 20)	2580

Total params: 29,351,508
Trainable params: 29,349,716
Non-trainable params: 1,792