Image Classification

Data 100/200A: Principles and Techniques of Data Science
Spring 2019

Linjian Ma, Lingyi Gu
EECS Department, University of California, Berkeley
{linjian, lingyigu}@berkeley.edu

Abstract

The objective of this project is to design and implement an image classification workflow based on the 1501 images provided. In this paper, we carefully document data preprocessing, exploratory data analysis, feature extraction, classifier training and assessment processes. Specifically, we carefully picked 22 features through EDA and careful visualization, and compared 5 different classifiers: logistic regression, K-nearest neighbor, Decision Tree, Random Forest and Support Vector Machine. The result shows that Logistic Regression classifier outperforms other classifiers, and achieve 42% in the validations set. In addition, we introduced a CNN-based neural network for comparison, whose results reach 100% accuracy in the training set and more than 45% accuracy in the validation set.

1 Introduction

Image classification has been increasingly popular with the thrift of deep learning, and it has become the benchmark of many machine learning algorithms. In this paper, we did extensive research over various traditional machine learning algorithms and try to test their effectiveness on image classification task on a small image dataset. We hope to answer the questions as follows:

- Which features are effective in the classification task?
- For the 5 classifiers we looked at, which one is more efficient after hyperparameter tuning?
- How will end-to-end deep learning with CNN architecture perform on this small dataset?

We will try to answer these questions in the following sections. The paper is organized as follows: at first we will introduce our data cleaning procedure, then we will introduce the features we chose in detail and show their effectiveness through data visualization. After that, we will talk about the training procedure with 5 different classifiers, and will also introduce the effectiveness of CNN based neural network.

2 Data Cleaning

Dataset The training set contains a total of 1501 RGB or grayscale images of different sizes (under folder `20_categories_training`). The testing set consists of a total of 716 images (under folder `20_validation`). In Figure 1, we show three example images of different categories in the training set. Each one has its own size / pixel intensity distributions. In Figure 2, we show the number of pictures in each category. As you can see, the `Gorilla` class consists of the most images. In addition, `Leopards`, `Penguin` also contain more images than the average.

Preprocessing Some of the input images are grayscale images, and we converted them to RGB images using `skimage.color.gray2rgb` (Pedregosa et al. (2011)) so that all images can be represented as 3-d arrays. The first two dimensions of the array correspond to the row and column pixels of the image. The third dimension corresponds to the RGB value, indicating the color intensity of each color channel.

Encoding The images are categorized into 20 different types. We extracted the type of images from the name of the folder using the following encoding: `0=Airplanes, 1=Bear, 2=Blimp, 3=Comet, 4=Crab, 5=Dog, 6=Dolphin, 7=Giraffe, 8=Goat, 9=Gorilla, 10=Kangaroo, 11=Killer-Whale, 12=Leopards, 13=Llama, 14=Penguin, 15=Porcupine, 16=Teddy-Bear, 17=Triceratops, 18=Unicorn, 19=Zebra`. We then created two dataframes for the training data, which contain the pictures and encodings series, and the test data, which contains only the pictures series, respectively.

Figure 1: Three Examples of Training Set Images

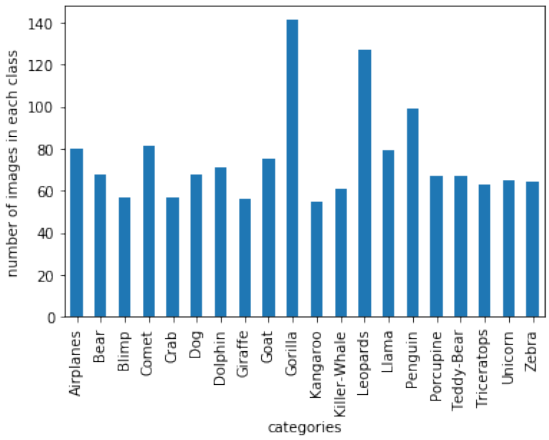


Figure 2: Category Frequencies

3 Feature Engineering and Exploratory Data Analysis

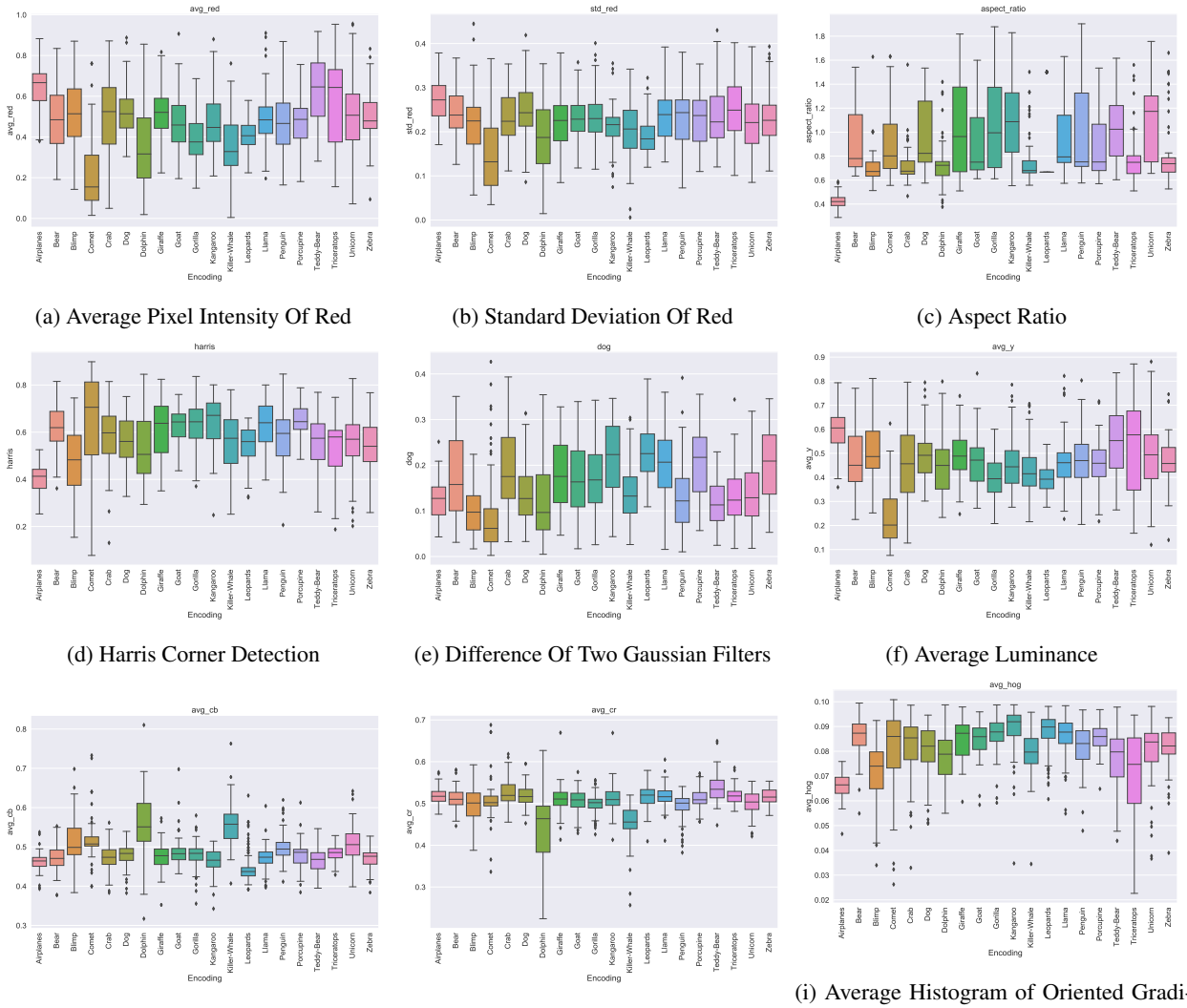
We categorize our features into 4 classes: the features related to the picture shape, pixel intensity, filtered information and leading signal. We introduce them separately as follows.

Picture shape We consider two features related to the shape of the picture: the picture size and the aspect ratio, defined as the ratio of the image height to the width. As can be seen in Figure 3, different categories have different aspect ratio distributions, making it a suitable feature. The distribution of size is shown in Figure 8, and the distribution difference for each category is not as obvious as aspect ratio feature.

Pixel intensity We investigate how the pixel intensity for each of the red, green, blue channel and grayscale images are varied between classes. For all of them, the `Comet` class has a lower pixel intensity than the average. This make sense because this type of images usually has a very dark background with a comet and some scattered stars. The distribution comparison for red is in Figure 3, and that for all the other colors are in Figure 8.

It is worth pointing out that the distributions of different colors don't differ too much (e.g. the average red distribution is similar to average green across different categories.)

We also investigate the features of luminance, blue chroma component as well as the red chroma component, in which the color information can be extracted from RGB intensities. As can be seen in Figure 3, different categories have diverse distributions, making these features suitable ones.



(a) Average Pixel Intensity Of Red

(b) Standard Deviation Of Red

(c) Aspect Ratio

(d) Harris Corner Detection

(e) Difference Of Two Gaussian Filters

(f) Average Luminance

(g) Average Blue Chroma Component

(h) Average Red Chroma Component

(i) Average Histogram of Oriented Gradients

Figure 3: Image Features Varying Between Classes

Filtered information We also consider the filtered information, meaning the latent patterns in the images that can only be extracted through some filters. We consider three features: amount of corners detected by Harris detection, the difference of images processed by two Gaussian filters, and the mean value of histogram of oriented gradients.

For Harris feature, it can be seen in the Figure 3 that except airplanes, blimp and comet, all the other categories have similar distribution. It is reasonable because many animals have similar shapes and therefore have similar number of corners. On the contrary, the dog (the difference of images processed by two Gaussian filters) and the hog (histogram of oriented gradients) features have various distributions over different categories.

Leading signal We also choose two matrix-based features: the leading signal amplitudes of the original image picked through the dot product between the image pixel intensities and leading singular vectors, denoting `svd`, and the leading signal amplitudes of the edge image picked by Canny edge detection, denoting `canny_svd`. We pick 5 leading amplitudes for `svd` feature and 2 leading amplitudes for `canny_svd` feature, based on the distribution check of amplitudes of different singular vectors.

Table 1: Features

Feature	Description
`size`	the size of the image normalized by dividing 1000000
`avg_red`	normalized mean value of the red channel image
`avg_green`	normalized mean value of the green channel image
`avg_blue`	normalized mean value of the blue channel image
`avg_gray`	normalized mean value of the grayscale image
`std_red`	normalized standard deviation of red channel
`std_green`	normalized standard deviation of green channel
`std_blue`	normalized standard deviation of blue channel
`std_grey`	normalized standard deviation of grayscale image
`aspect_ratio`	the height to width ratio
`harris`	amount of corners detected by Harris corner detector (Harris et al. (1988))
`dog`	Differences of images processed by two Gaussian filters with variance 0.3 and 0.5
`avg_y`	normalized mean value of luminance
`avg_cb`	normalized mean value of blue chroma component
`avg_cr`	normalized mean value of red chroma component
`std_y`	normalized standard deviation of luminance
`std_cb`	normalized standard deviation of blue chroma component
`std_cr`	normalized standard deviation of red chroma component
`avg_hog`	mean value of Histogram of Oriented Gradients (Dalal and Triggs (2005))
`std_hog`	standard deviation of Histogram of Oriented Gradients
`svd`	dot products of vectorized pixel intensities with leading singular vectors of the whole image dataset
`canny_svd`	dot products of canny edge image pixels with leading singular vectors of edge images (Canny (1987))

4 Classifier Training

We tried the following 5 machine learning models on the learning set and tuned the hyper-parameters for the best predictions. We used cross-validation to evaluate these models. For each experiment in the cross-validation procedure, the learning set was splitted into a training set (80% of the data) and a validation set (20% of the data).

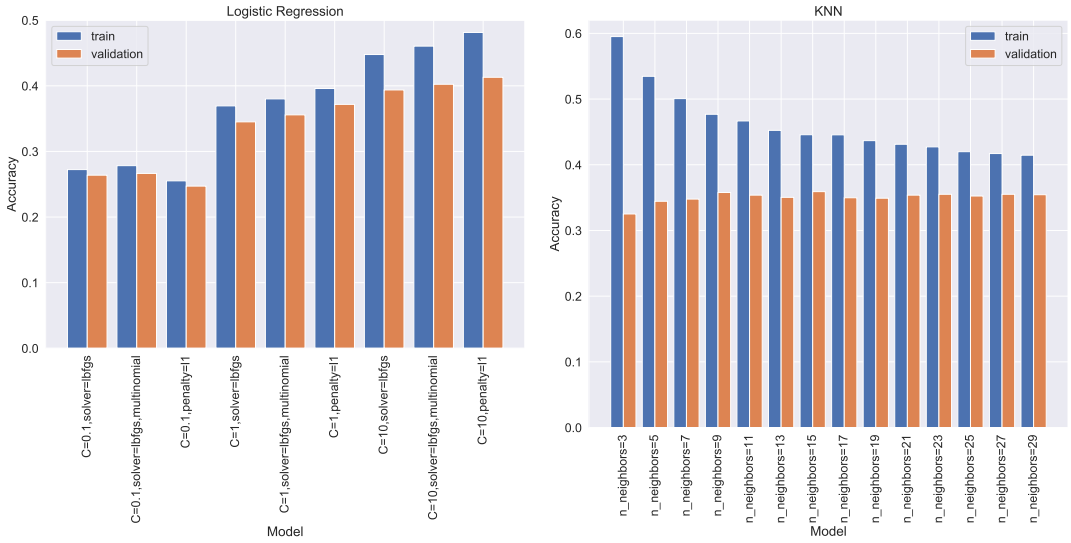
- Logistic Regression: sklearn.linear_model.LogisticRegression
- K-nearest Neighbors: sklearn.neighbors.KNeighborsClassifier
- Classification Tree: sklearn.tree.DecisionTreeClassifier
- Random Forests: sklearn.ensemble.RandomForestClassifier
- Support Vector Machine (SVM): sklearn.svm.SVC

Logistic Regression The first model we tested was the basic logistic regression model. `penalty` is used to specify the norm used, which can be $L1$ or $L2$. `solver` is the algorithm to be used in the optimization problem, where `"lbfgs"` solver only supports $L2$ norm. Moreover, we wanted to add a regularization parameter `C`, representing the inverse of regularization strength, to avoid over-fitting.

Figure 4a shows the accuracy score (on a scale of 0 to 1) for different penalization norm $L1$ or $L2$, solver and regularization parameter. The accuracy on both the training and validation set does not change much on different norms and solvers, which means they have a small impact on the result. However, A larger `C` increases the prediction accuracy. The best validation accuracy is 41.3% with `C=10, penalty="11", solver="lbfgs", multi_class="warm"`.

K-nearest Neighbors The K-nearest neighbors (KNN) classifier implements a k-nearest neighbors vote. In this function, the `n_neighbors` parameter represents the number of neighbors k to be used for k neighbors queries. We searched for the best model by setting `n_neighbors` to every other number in $[3, 30)$.

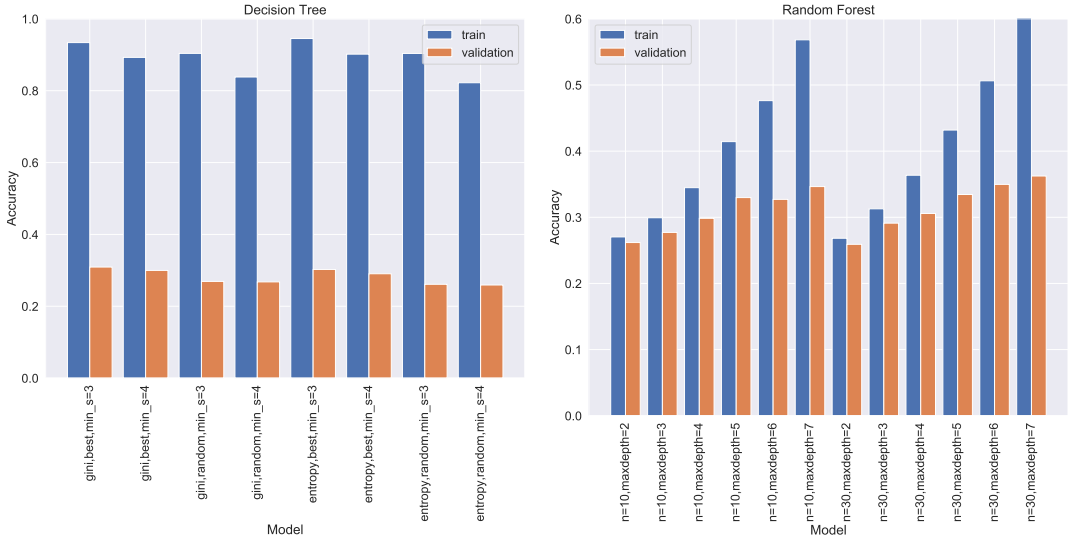
Figure 4b shows the accuracy score (on a scale of 0 to 1) for different value of k. The training accuracy decreases as k decreases, and validation accuracy is mostly in between 0.3 and 0.4 for varied ks. The best validation accuracy is 35.9% with `n_neighbors=15`.



(a) Accuracy of Logistic Regression

(b) Accuracy of K-nearest Neighbors

Figure 4



(a) Accuracy of Decision Tree Classifier

(b) Accuracy of Random Forest

Figure 5

Decision Tree The parameters we tested are `criterion`, `splitter` and `min_samples_split`. `criterion` is the function to measure the quality of a split. It takes in either `"gini"` (Gini impurity) or `"entropy"` (information gain). `splitter` is the strategy used to choose the split at each node, in which `"best"` chooses the best split and `"random"` chooses the best random split. `min_samples_split` is the minimum number of samples required to split an internal node.

Figure 5a shows the accuracy score (on a scale of 0 to 1) for varied `criterion`, `splitter` and `min_samples_split`. The best prediction result have an accuracy of 30.9% with `criterion="gini", splitter="best", min_samples_split=3`. It is worth noticing that the predication accuracy on the train set is very high while the accuracy on the validation set is much lower. It is obvious that overfitting occurs easily using the decision tree classifier, and we need to find something that improves it.

Random Forest The random forest classifier fits a number of decision tree classifiers on sub-samples of the data to improve the predictive accuracy. We tested the `n_estimators`, the number of trees in the forests, as well as `max_depth`, the maximum number of levels in each decision tree.

Figure 5b shows the accuracy score (on a scale of 0 to 1) for different numbers of `n_estimators` and `max_depth`. The best prediction result have an accuracy of 36.2% with `n_estimators=30, max_depth=7`. We can also see that the validation accuracy increases a little bit, comparing the result of random forest to decision tree. The training accuracy is much lower compared to the decision tree classifier, which means that the overfitting problem is eased.

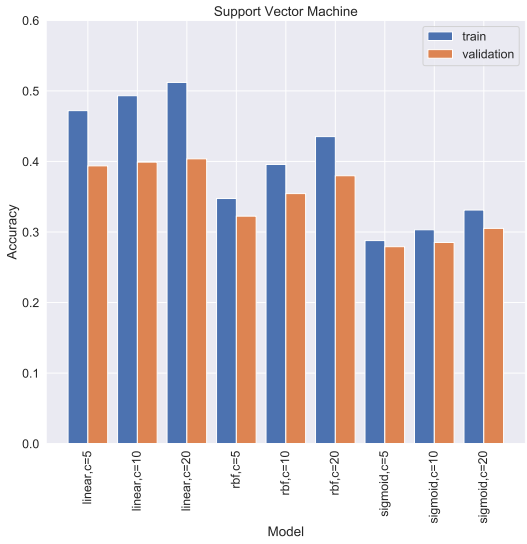


Figure 6: Accuracy of Support Vector Machine

Support Vector Machine Support Vector Machine (SVM) finds the best hyperplane to separate different classes maximizing the distances between the points and the plane. We tested the algorithm using different `kernel` type: `"linear"`, `"rbf"` and `"sigmoid"` with different penalty parameter `C` for the error term.

Figure 6 shows the accuracy score (on a scale of 0 to 1) for different `kernal` function with varied `C`. The best prediction result has an accuracy of 40.3%, where `kernel="linear", C=20`.

Table 2: Prediction Accuracy (Learning Set)

Classifier	Accuracy	Parameters
Logistic Regression	41.3%	`C=10, penalty="11", solver="lbfgs", multi_class="warm"`
K-Nearest Neighbors	35.9%	`n_neighbors=15`
Decision Tree	30.9%	`criterion="gini", splitter="best", min_samples_split=3`
Random Forest	36.2%	`n_estimators=30, max_depth=7`
Support Vector Machine	40.3%	`kernel="linear", C=20`

Performance Assessment By hand picking features from the images, we were able to achieve an accuracy of 41.3% by classifier training. Among all classifiers (figure 2), Logistic Regression has the best prediction result. We then generated the same 22 features for the validation images (under folder `20_validation`) and ran the our best model on it to get the image class predictions.

5 Neural Network

We used Alexnet (Krizhevsky et al. (2012)) like model to train the dataset using PyTorch (Paszke et al. (2017)). Specifically, the model has two convolutional layers, each followed by a nonlinear ReLU function, batch normalization layer and max polling layer. After them, three fully connected layers are used, first two of which are followed by a nonlinear ReLU function.

To train our dataset with different image sizes on the structure, we resize all the images into shape 32×32 with the `cv2.resize` function developed by OpenCV. In addition, we normalize all the images in each color channel so that it has mean values approximately 0 and variances 1. We trained our model with Stochastic Gradient Descent (SGD), with the batch size set as 32, learning rate set to be 0.01, momentum set to be 0.9. We use weight decay (Krogh and Hertz (1992)) to regularize the training, whose value is set to be $5e-4$. We trained the model until 100 epochs, at which time the training losses are almost 0.

The training curves are in Figure 7. We can see that after 20 epochs, the training loss is almost equal to 0, meaning that it has fully converged. After 20 epochs, the validation accuracy reach the level that is almost higher than 0.45. It is slightly better than the best accuracy from last section (41.3%), and shows the advantage of end-to-end deep learning algorithms.

It is worth mentioning that the reason our final validation accuracy (higher than 45%) is not as good as state-of-art results (97% for MNIST, 94% for CIFAR10 and around 80% for ImageNet) is that the dataset is too small. In this case, it will be very easy for the neural network to overfit and not generalize well.

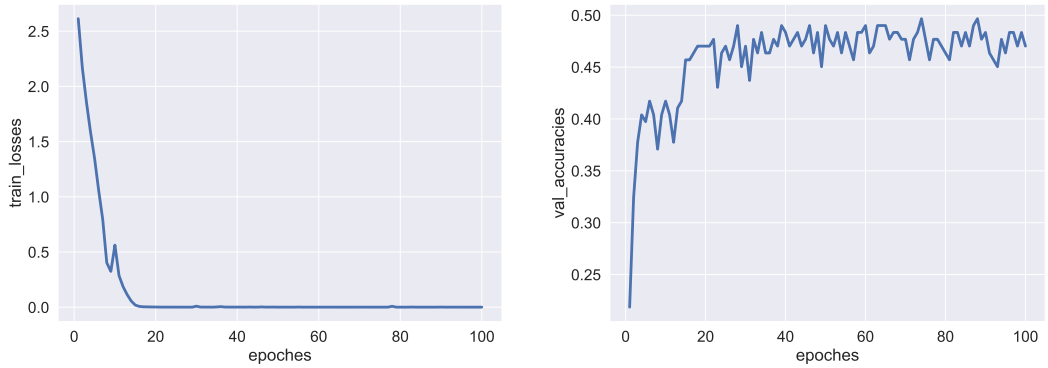


Figure 7

6 Conclusion

Multi-class image classification is possible with machine learning models and neural network classifiers. Using the traditional feature picking method, we are able to reach a prediction accuracy of 41.3%. It is a decent score, since we have 20 different categories but only a small dataset of about 1500 images. There are several factors we consider important in this classification process. Firstly, exploratory data analysis is crucial for feature selections. We make sure the features are representative enough as they should vary between different categories in order to obtain the characteristics of each individual category. This has a direct impact on the prediction. Secondly, tuning the hyper-parameters is necessary. In this project, we evaluate the performance of different combination of hyper-parameters. The highest accuracy score we get is above 40% while the lowest score is below 30%, a pretty big difference. Moreover, cross-validation is another crucial step, especially when we have a small dataset. In our case, we have 1501 training images with 20 categories. We spot some over-fittings when we evaluate the performance on one set of training and validation splits. We get a high 90s in the training accuracy whereas the validation accuracy is around 30s on the decision tree classifier. Finally, neural network produce a slightly higher score than traditional machine learning

algorithms. However, it is also easy for neural network to overfit on a small dataset like what we have. We need a larger dataset multi-class image classification problems in order for neural networks to performance better. Extracting features from the last layer of the neural networks and passing them into to a machine learning model might also improve the predictive accuracy.

References

John Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE Computer Society, 2005.

Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. Citeseer, 1988.

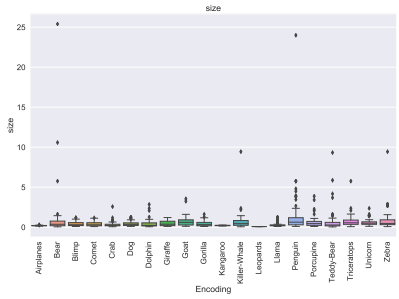
Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

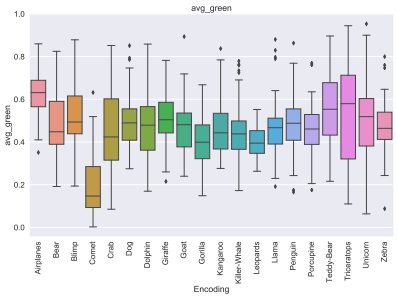
Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration*, 6, 2017.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

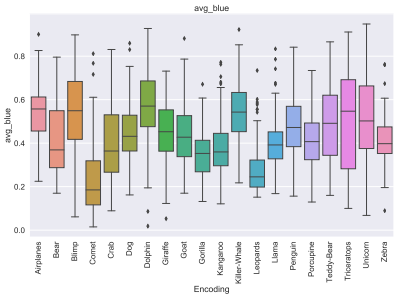
7 Appendix



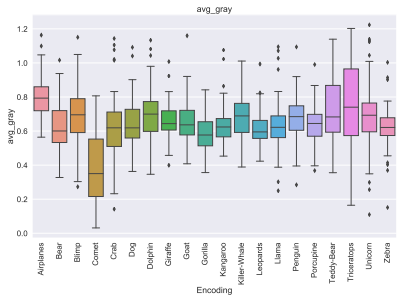
(a) Size



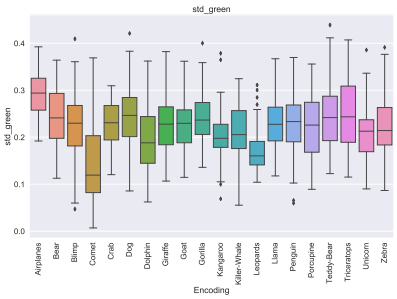
(b) Average Pixel Intensity Of Green



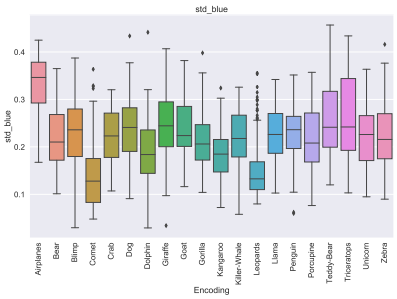
(c) Average Pixel Intensity Of Blue



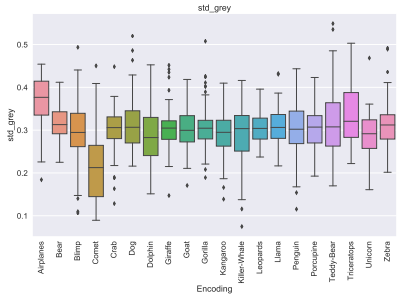
(d) Average Pixel Intensity Of Grayscale



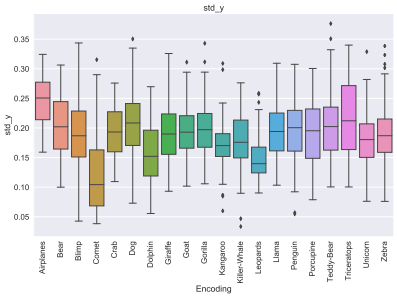
(e) Standard Deviation Of Green



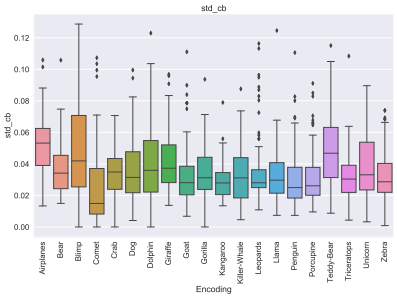
(f) Standard Deviation Of Blue



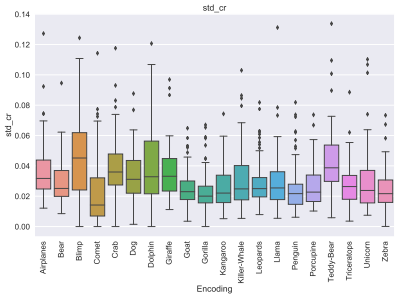
(g) Standard Deviation Of Grayscale



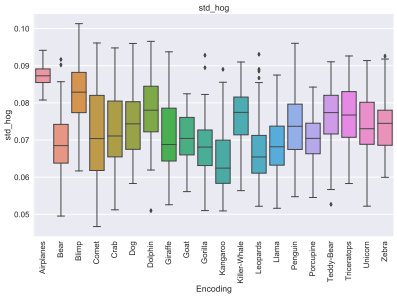
(h) Standard Deviation of Luminance



(i) Standard Deviation Of Blue Chroma Component



(j) Standard Deviation Of Red Chroma Component



(k) Standard Deviation Of Histogram of Oriented Gradients

Figure 8: Image Features Varying Between Classes