Extended CS200A Final Project: Image Detection

December 13, 2019

Emeri Zhang

Abstract:

In this project, I utilize the entire data science lifecycle on an image classification problem, performing data input and data processing, exploratory data analysis and feature extraction, classifier training, and a classifier performance assessment. I utilize this as a learning experience to explore feature extraction algorithms with OpenCV, then detail the feature selection process by visualizing distributions over each class. I train and parameter tune various sklearn-based models including support vector machine, k-nearest neighbor, logistic regression, classification trees, and random forest. The SVM model performed the best with an accuracy of 0.4286, followed by Random Forest with an accuracy of 0.4252. We concluded that corner features and key points were good features, while some models like BRIEF performed better than ORB. Finally, I introduce a CNN-based neural network for comparison, where we find an interesting correlation between regularization and the model's generalizability.

Implementation:

Library versions: cv2 version 3.4.2, scikitlearn 0.21.3, pandas 0.25.3, skimage 0.15.0

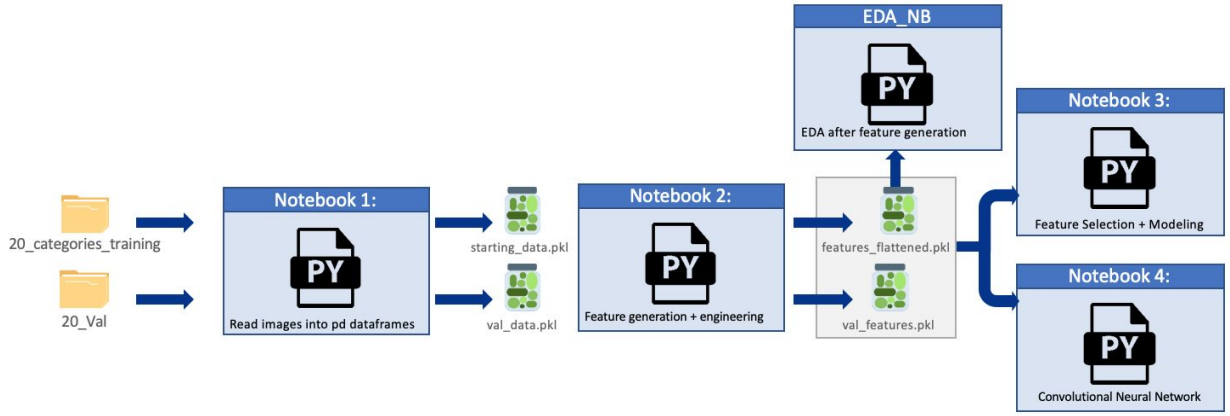


Figure 1: *File Paths to show the different notebooks. Notebook 1 is where the images are read into a dataframe, Notebook 2 is for feature generation and engineering, EDA_NB is for Explanatory Data Analysis, Notebook 3 is for Feature Selection and Modelling, and Notebook 4 is for Neural Networks*

Data Ingestion and Preprocessing: GradProject_NB1.ipynb

I was provided a dataset of 1501 images of 20 classes. I read these into a dataframe of shape 1501, 2 with column **image** containing a 3-d array, with dimensions row pixels, column pixels, RGB color intensity (between 0 and 28 − 1). Column **label** is the class label encoded alphabetically, with the following mapping:

0	airplanes		10	kangaroo
1	bear		11	killer-whale
2	blimp		12	leopards
3	comet		13	llama
4	crab		14	penguin
5	dog		15	porcupine
6	dolphin		16	teddy-bear
7	giraffe		17	triceratops
8	goat		18	unicorn
9	gorilla		19	zebra

Table 1: *Shows the different images, and the label number for each class of image*

I note 15 images are grayscale and lack RGB channels and thus we see a dimension mismatch with representation in a 2-D array. 12 of these images are class 3, or comet, making up 15% of the comet training images -- too many to drop, so I impute missing RGB channels with cv2.

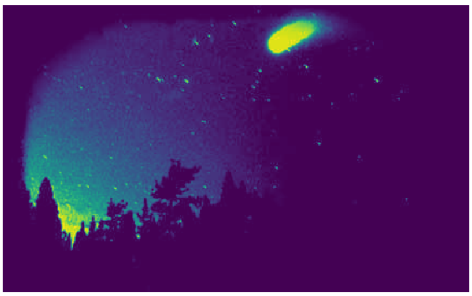
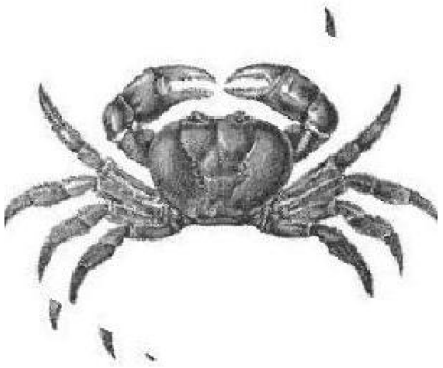
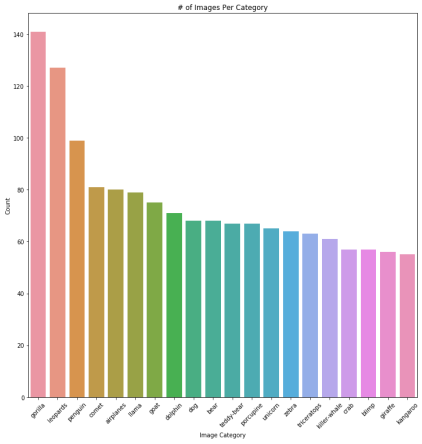


Figure 2: *LEFT: original comet image missing RBG channel*
RIGHT: comet image visualized after RGB values imputed
LEFT: original crab image missing RBG channel
RIGHT: crab image visualized after RGB values imputed



Feature Extraction and Engineering: GradProject_NB2.ipynb

As a cursory exploration, I view **1)** image count over classes to identify any imbalances, and **2)** average image size (# of pixels). I note **1)** gorillas, leopards, and penguins dominate the set, while remaining classes have roughly uniform distribution, and **2)** penguins, bears, and zebras images are largest



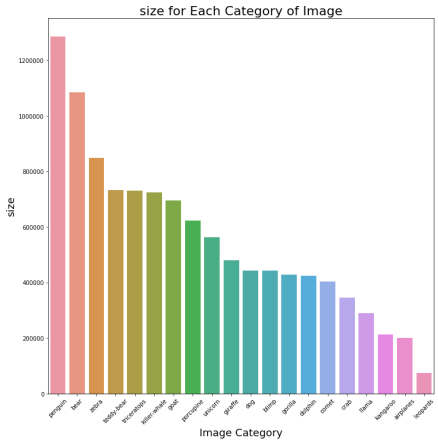


Figure 3: *LEFT: number of images in each category*
RIGHT: image size in each category

Note: For manual feature extraction and modeling with sklearn, I decided size is an important feature. I see some feature detection algorithms are not scale or rotation-invariant, and choose not to normalize or standardize orientation or scaling.

Based on literature and documentation on openCV, I selected 25 features.

Features fall in 3 domains: 1) color channels 2) corner and edge detection 3) key point detection

Feature Name	Description
size	Total pixel size
aspect_ratio	aspect ratio of the image
r_mean	Mean for the red channel intensity
r_std	Standard deviation pixel value for red channels
g_mean	Mean for the green channel intensity
g_std	Standard deviation pixel value for green channels
b_mean	Mean for the blue channel intensity
b_std	Standard deviation pixel value for blue channels
luminance_mean	Mean of luminance (Y)
luminance_std	SD of luminance (Y)
cb_mean	Mean of blue chroma component (Cb)
cb_std	SD of blue chroma component (Cb)
cr_mean	Mean of red chroma component (Cr)
cr_std	SD of red chroma component (Cr)
brief	BRIEF (Binary Robust Independent Elementary Features), returns # of key points
orb	ORB (Oriented FAST and Rotated BRIEF), returns # of key points
sift_kp	SIFT (Scale Invariant Feature Transform), returns # of key points
surf_kp	SURF (Speeded-Up Robust Features), returns # of key points
fast_corner	FAST Algorithm for Corner Detection, returns # of key points
canny_edges	Canny edge detector that returns the edge gradient; sum
prewitt_h	Prewitt operator for finding edges in the horizontal direction; sum
prewitt_v	Prewitt operator for finding edges in the vertical direction; sum
binarize	Binarization of grayscale image, returns matrix of 0 or 1; mean
harris	Harris corner detector returns a matrix for the corners extracted; mean
shi_tomasi	Shi Tomasi corner detector returns a matrix for the corners extracted; mean
label	Class labels, encoded 0-19

Table 2: *Feature metadata, shows the 25 different features extracted and color coded on the domain it falls under*

Color Channels: Upon a quick skim through the image files, I note some classes have different prevalent colors i.e. comets lack RGB channels, dolphins and whales are more dominated by the blue water, unicorns have a more "sparkly" property. Thus, I decided to calculate the mean and standard deviation for channel values, and do the same for Y (luminance), Cb (chroma blue), and Cr (chroma red). Note I also binarized a grayscale version of the image, in case it is useful to eliminate the concept of color altogether and revert to grayscale only.

Corner and Edge Detection: When we perceive images, our brain processes colors and shapes. A first instinct was to understand how to distinguish boundaries on images, and after reading literature, I understood that from a computer's perception of an image, an edge is where there is a change in color, and corners are point-like features of images that are specific points of interest. There are many corner and edge detection algorithms available, and I chose a few commonly used ones from openCV. Below, we see canny edges with various sigma values, and I chose sigma as 2.

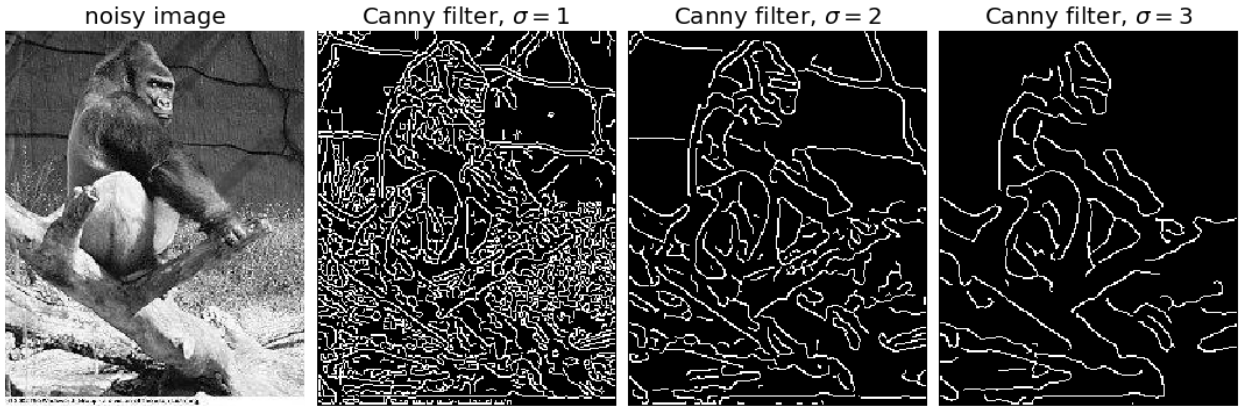


Figure 4: *Canny images shown for different sigma values*

Note: Several feature detection algorithms yielded multi-index arrays of varying sizes, which is not handled when modeling in scikitlearn. To ensure feature matrices align for all images in the learning set, I flattened the features and summarized as best fit.

For edge detection features, I took the sum of flattened arrays due to the sparsity of the feature matrix. For corner detection features, I chose mean due to denser feature matrices.

Key Point Detection: Besides corners, other interesting points exist as spatial locations or points on an image, which might define areas that help a specific class stand out and thus be more easily identifiable. Below we see an original test image, followed by plots of key points from BRIEF, ORB, and SIFT key point detection algorithms.

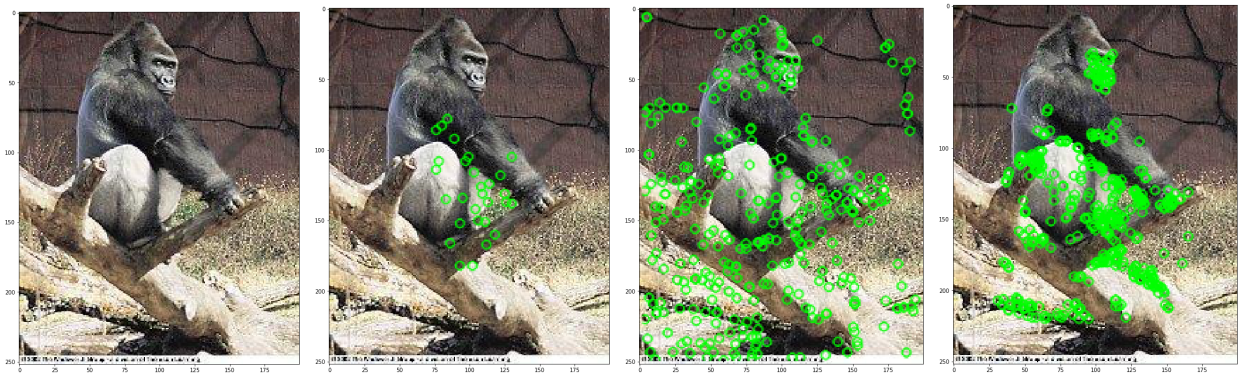


Figure 5: *FIRST: original test image SECOND: key points from BRIEF THIRD: key points from ORB FOURTH: key points for SIFT*

For key point detection features, I returned the length of the key points array, yielding a scalar number of "important points" as the feature.

For Fun: "Fast & Furious Presents: Features": GradProject_NB2.ipynb

In selecting features, a more in depth discussion of rationale for choosing algorithms follows. Some algorithms have been shown to be faster than others, while others show "comparable" results. As a learning exercise, I am curious to see:

1. **Time test** of the algorithms to test these time test claims
2. **Dimensionality reduction** to see if after my summarizing of the feature matrices, how speed affects feature importance

Purpose of the exercise is to:

1. Deeper dive into algorithm functionality and understanding
2. **Why not!**

SIFT (Scale Invariant Feature Transform):

SIFT (Scale Invariant Feature Transform), developed in 2004, is an image descriptor for image-based matching and recognition invariant to image scale and rotation.

There are 5 key steps:

1. **Scale-space Extrema Detection:** Real world objects are meaningful only at a certain scale; we find the proper scale with scale-space filtering. Difference of Gaussians is found, acting as a blob detector for various sizes due to change in sigma. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different sigma values. Once this DoG are found, images are searched for local extrema, which are potential keypoints.
2. **Keypoint Localization:** From the previous steps, some potential key points lie along an edge, or they don't have enough contrast, and we do not find these useful. Taylor series expansion is used to get more accurate location of extrema. If the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. DoG has higher response for edges, so edges need to be removed. Eliminate any low-contrast key points and edge key points to get strong interest points.
3. **Orientation Assignment:** An orientation is assigned to each keypoint to achieve invariance to image rotation. A neighborhood around each keypoint is taken, and gradient magnitude and direction are calculated. A weighted orientation histogram is created. The highest peak in the histogram, and any peak above 80% is also considered to calculate the orientation.
4. **Keypoint Descriptor:** A 16x16 neighbourhood around the keypoint is taken, with 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. 128 bin values are available. It is represented as a vector to form keypoint descriptor.
5. **Keypoint Matching:** Keypoints between two images are matched by identifying their nearest neighbours

SURF (Speeded-Up Robust Features):

SURF has been shown to be 3x faster than SIFT, with comparable performance. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change. From EDA, there is some viewpoint and illumination change, so this may be an interesting feature.

Brief (Binary Robust Independent Elementary Features):

SIFT uses 128-dim vector for descriptors, or 512 bytes, as it uses floating point numbers, and SURF also takes minimum of 256 bytes (for 64-dim). Creating a vector for descriptors for thousands of features takes a lot of memory, which is not as reasonable with resource constraints, and takes longer. BRIEF finds binary strings but skips the step of finding descriptors and thus large vectors. It takes smoothened image patches and selects a set of n_d (x,y) location pairs in a unique way, then performs some pixel intensity comparisons on these location pairs. ***BRIEF is a faster method than SIFT and SURF feature descriptor calculation and matching, with high recognition rate. Let's see how they compare, speedwise.***

ORB (Oriented FAST and Rotated BRIEF):

ORB is a fusion of FAST keypoint detector and BRIEF descriptor, with modifications for performance.

1. **First it uses FAST to find keypoints**
2. **Harris corner measure to find top N points among them.**

3. **Pyramid to produce multiscale-features.**

ORB is much faster than SURF and SIFT and ORB descriptor works better than SURF. ORB is a good choice in low-power devices, us in this case.

FAST Algorithm for Corner Detection:

High-speed performance. Not robust to high levels of noise. It is dependant on a threshold.

1. Select a pixel **p** in the image which is to be identified as an interest point or not. Find its intensity, $\mathbf{I_p}$.
2. Select appropriate threshold value, **t**.
3. Consider a circle of 16 pixels around the pixel.
4. p is a corner if there exists a set of n contiguous pixels in the circle (of 16 pixels) which are all brighter than $\mathbf{I_p + t}$, or all darker than $\mathbf{I_p - t}$
5. For speed,first compare the intensity of pixels at cardinal directions of the circle around $\mathbf{I_p}$.
6. If at least three of the pixels are above or below $\mathbf{I_p + t}$, then check for all 16 pixels and check if 12 contiguous pixels fall in the criterion.
    a. Else, not an interest point.

For detecting multiple interest points in adjacent locations, I use non-maximum suppression.

Canny Edge Detector:

Claimed to be “probably the most commonly used and most effective edge detection method”

1. **Noise Reduction:** Reduce noise with a 5x5 Gaussian filter
2. **Finding Intensity Gradient of the Image:** Filter with a Sobel or Prewit kernel in both horizontal and vertical direction to get first derivatives, used to compute edge gradient magnitude and direction, per pixel.
3. **Extract Edge Points with Non-maximum Suppression:** Full scan of image to remove any unwanted pixels which may not constitute the edge. Every pixel is checked if it is a local maximum in its neighborhood in the direction of the gradient. If not, it is . This step results in a binary image with "thin edges"
4. **Linking and Thresholding with Hysteresis**: decide which are all edges are really edges and which are not. Set a minVal and maxVal threshold. Any edge with:
    a. Intensity gradient >maxVal are sure to be edges
    b. <minVal are discarded as non-edges
    c. minVal<edge<maxVal: If connected to “sure-edge” pixels, edges. Else, discard.

Also removes small pixels noises on the assumption that edges are long lines. **Final result:** strong edges of image

Prewitt Operator:

Prewitt is used for detecting vertical and horizontal edges in images, without placing emphasis on the pixels that are closer to the center of the mask.

1. Convolve the image with a small, separable, and integer valued filter, providing differentiating (for edge response) and smoothing (reducing noise).

Harris Corner Detector:

Corners are regions in the image with large variation in intensity in all directions.
There are 3 cases in change of intensity:

1. **Flat region**: no change in all directions
2. **Edge**: no change along edge directions
3. **Corner**: significant change in all directions

Harris measures change of intensity for shifts in windows. For desired very distinctive patches, this will be larger; we maximize the change of intensity function using Taylor Expansion, yielding a matrix form score equation. Eigenvalues are used to determine if the region is corner, edge or flat. Provided the image with eigenvalues, we threshold to yield corners.

Shi Tomasi Corner Detector:

Shi Tomasi: modified and improved Harris Corner Detection.

Based on literature, we have the following algorithms summary:

In terms of speed: BRIEF *faster than* SURF *3x faster than* SIFT FAST keypoint detector + BRIEF descriptor = ORB Canny Edge Detector: probably the most commonly used and most effective edge detection method Corners: Shi Tomasi > Harris

Table 3: *Feature extraction algorithms summary*

We see BRIEF is 9.64578x faster than SURF, and SURF is 1.877x faster than SIFT. Shi Tomasi is ~22% faster than Harris.

Dimensionality Reduction and EDA with PCA and t-SNE:

Now, we explore feature importances. PCA is a linear feature extraction method, which aims to combine features in a way allowing us to discard the less important features while retaining the most valuable, or explanatory parts, of the features. Each new component is independent from the others. We plot the number of components parameter vs the cumulative explained variance, and choose n_components = 15. Below, we see this plot. We also visualize in 2D principal component 1 on the x-axis and principal component 2 on the x-axis, with category labels at the median x,y position. PC1 and PC2 explain ~45% variance.

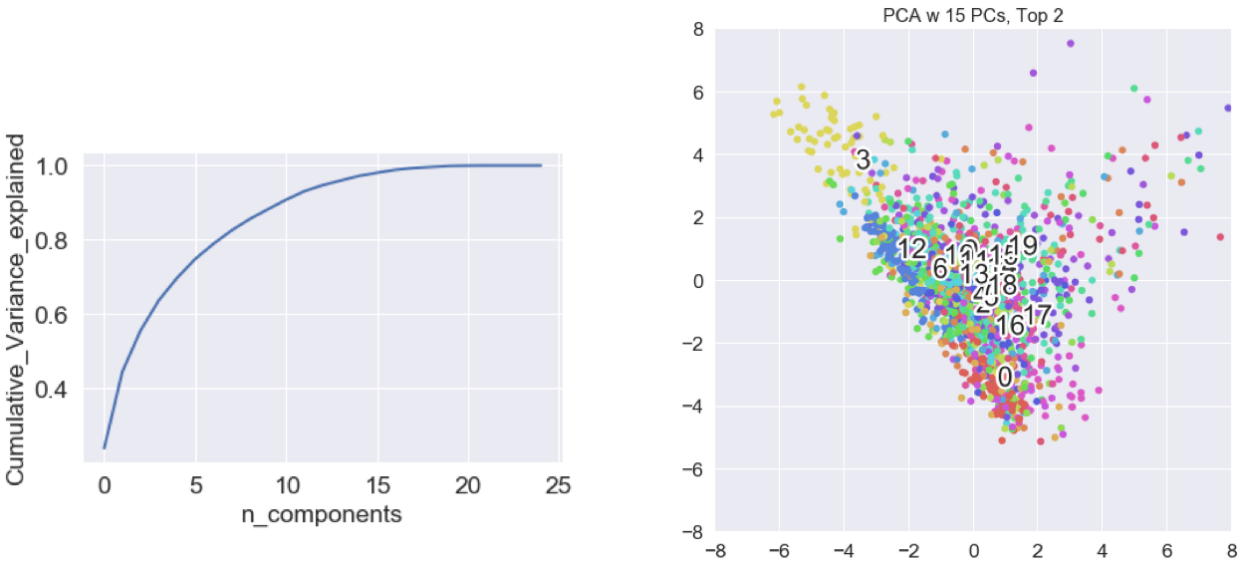


Figure 6: *LEFT: n_components vs Cumulative Variance Explained* *RIGHT: PC 1 vs PC2*

This PCA with 15 Components explains **~97%** of the data's variance. PCA has tried to separate points and form Clusters of similar points. From a 2D plot, we don't see very clear clusters, which is expected as we are summarizing 15 dimensions in a 2D visualization. We may use the principal components later for feature selection.

Next, to yield a better visualization for the sake of data exploration and build intuition behind the data, we try t-SNE, t-Distributed stochastic neighbor embedding to minimize the divergence between two distributions with Kullback-Leibler divergence with gradient descent. t-SNE is incredibly flexible, often finding structure where other dimensionality-reduction algorithms may not be able to do so, ut this flexibility lends itself to interpretation complexity.

Essentially, it minimizes divergence between two distributions: **1)** distribution that measures pairwise similarities of the inputs, and **2)** distribution measuring pairwise similarities of corresponding low-dimensional points in the

embedding. The goal is to take the set of points in high-dimensional space and map it to a lower dimensional space, attempting to find patterns in the data by identifying observed clusters based on similarity of data points with multiple features. t-SNE is non-linear, adapting to underlying data, and performing different transformations on different regions. We must also tune the "perplexity" parameter, which provides some balance between local vs global factors in the data, and essentially guesses the number of close neighbors per point. We will look at multiple perplexity values to provide the most complete picture, and provide multiple runs to converge on a more stable global shape.

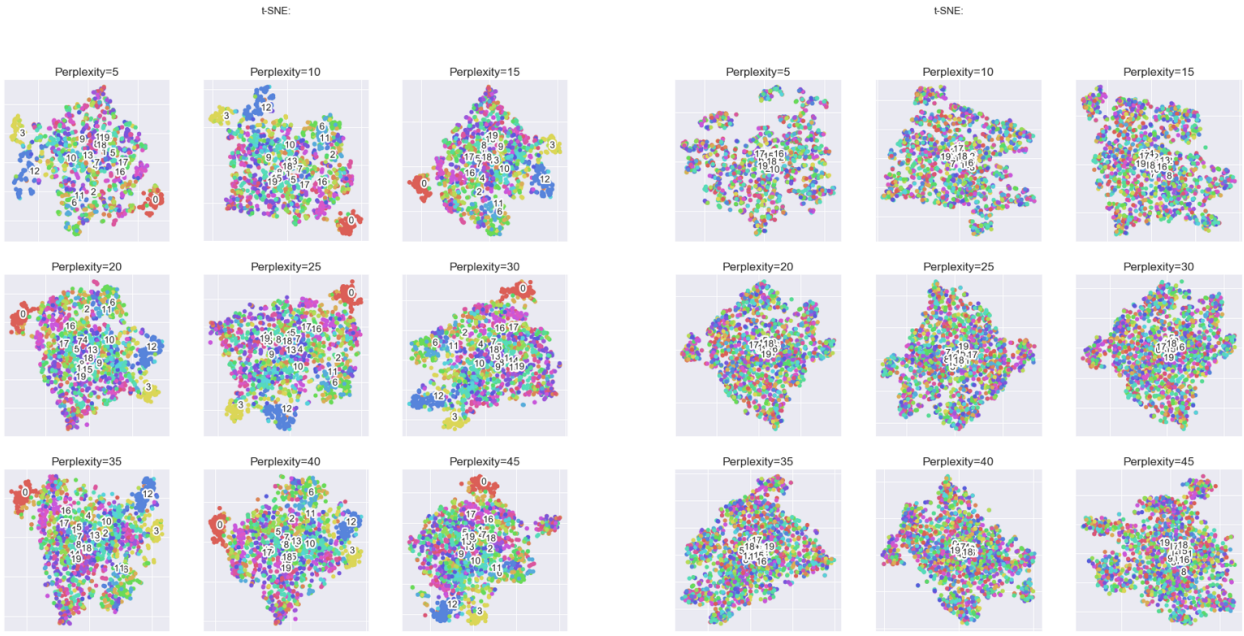


Figure 7: *LEFT: t-SNE with varying perplexities, with original data* *RIGHT: t-SNE with varying perplexities, run post PCA*

On the left, we see t-SNE run with complexities 5 to 45, other parameters held constant. The separation is not particularly clear among any of the perplexity levels, but **groups 0, 3, 12, which are airplanes, comets, and leopards respectively, persistently show clear clusters.**

On the right, we repeat this exercise, but on the data post PCA with 15 principal components. The results yielded show significantly worse separation - this may hint that the feature combinations that help distinguish categories 0, 3, and 12 and included in the PCs >15.

Finally, below, we compare PCA results from above with the best t-SNE visualization.

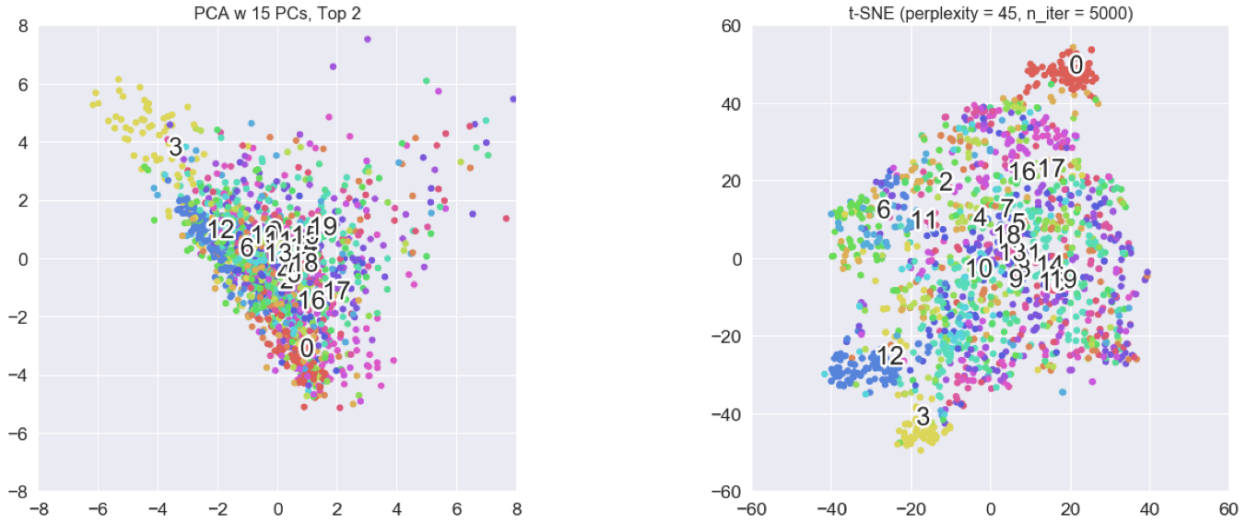


Figure 8: *LEFT: PC 1 vs PC2, PCA with 15 PCst-SNE with 15 components,* *RIGHT: t-SNE perplexity 45, and 5000 iterations, no PCA*

EDA: EDA_NB.ipynb

After generating the feature frame, we created visualizations to explore features across the 20 classes.

1) We plotted seaborn barplot of the **means in descending order** to get a general summarized idea of which classes have smaller or larger feature values
2) We created boxplots to **identify quartiles and observe any outliers** across classes.
3) We created violin plots **exploring distribution and variance and skew across classes**.

Some interesting findings are summarized below:

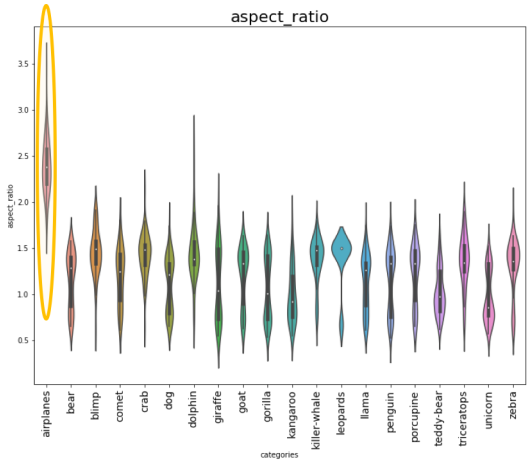


Figure 9: *Aspect ratio across categories*

The image above depicts that airplanes have a higher aspect ratio compared to the aspect ratios of the other labels.

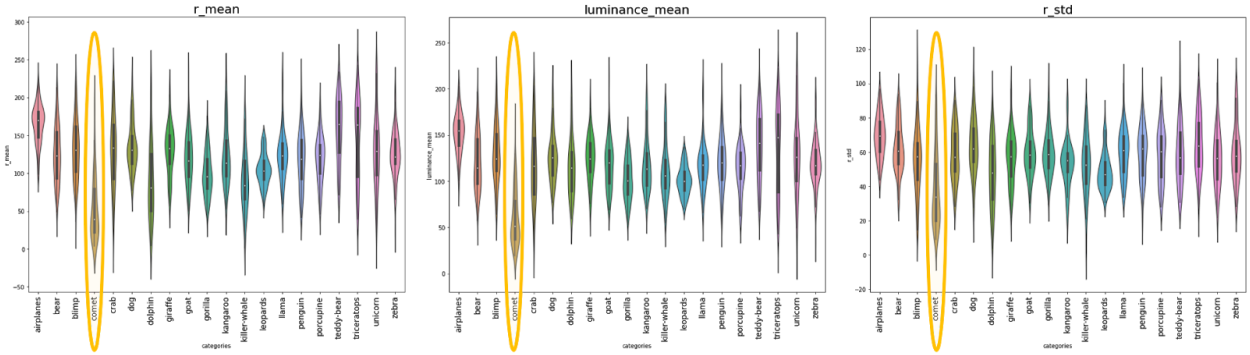


Figure 10: *LEFT: red channel mean* *MIDDLE: red channel std* *RIGHT: luminance mean*

The three images above show that comets have a low **r_mean**, low **r_std**, low **luminance_mean**. We also note (not comets have a low g_mean, while comets and leopards also have a low b_mean and a low b_std.

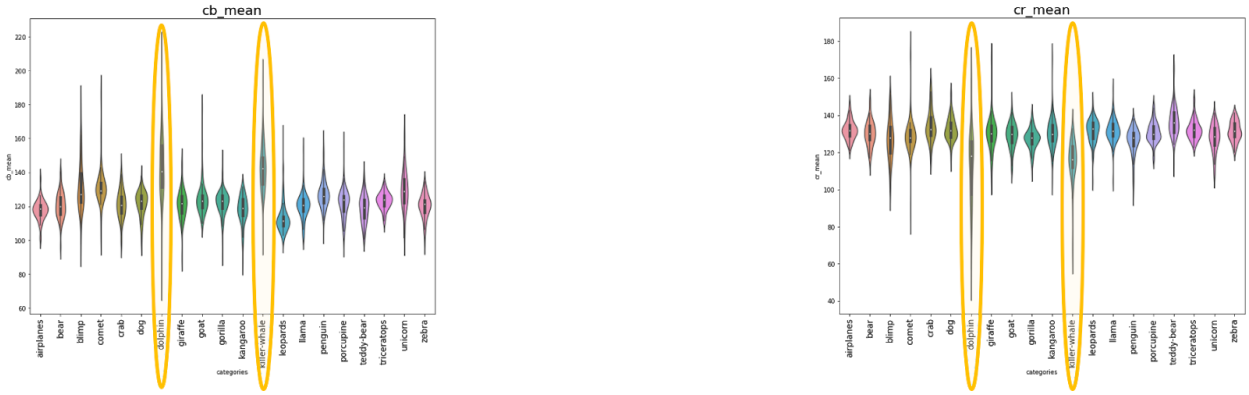


Figure 11: *LEFT: blue chroma component mean across all labels* *RIGHT: red chroma mean across all labels*

Dolphins and killer-whales tend to have higher **cb-mean** and lower **cr-means**. This makes sense because dolphins and whales are aquatic creatures, and these images should have a heavier blue tone because they are taken in the water.

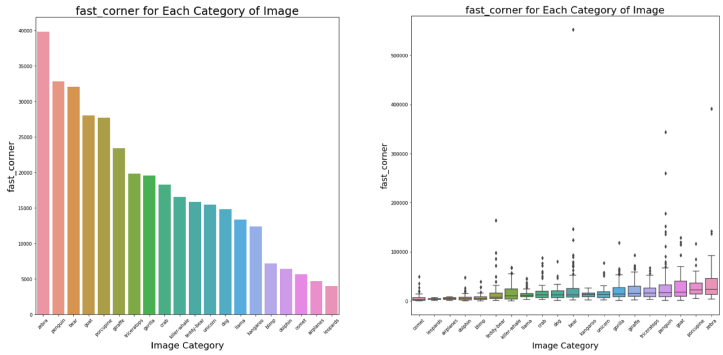


Figure 12*: Number of keypoints across all labels from Fast_Corner method*

The image above shows the number of key points for each label from using the FAST algorithm in corner detection. There is a clear variation in means among the different labels. However, upon a closer look at the boxplots, we note this is largely due to outliers, and the medians are relatively similar.

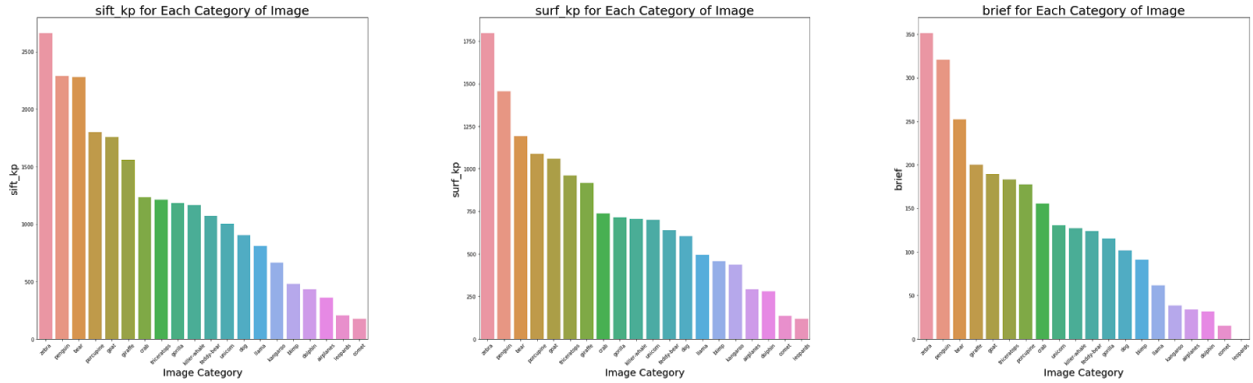


Figure 13: *# keypoints LEFT: SURF* *MIDDLE: SIFT method* *RIGHT: BRIEF method*

The image above shows the number of key points for each label from using the SIFT, SURF and BRIEF feature. Similar to the Fast_corner feature, there is a clear trend where each label returns a specific range of keypoints.

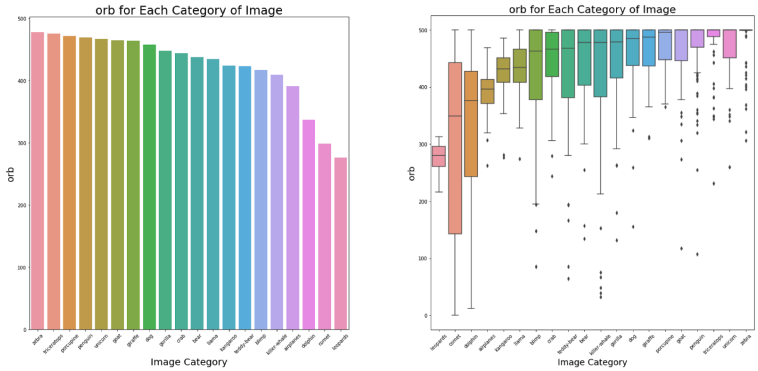


Figure 14: *Number of keypoints returned by ORB*

The image above shows the ORB feature that returns the number of keypoints for each of the labels. ORB is a more efficient alternative to SIFT or SURF for detecting keypoints, and we were expecting the histogram to follow a similar trend to the SIFT, SURF and BRIEF features. However, these trends seem to be more incremental and less distinguishable. There are fewer keypoints, which can be seen from the figures under Keypoint Detection.

Modeling: GradProject_NB3.ipynb

We four models on the dataset, all with 5 fold cross-validation, also tuned parameters to get the best performance. In this section, we will walk through each model. Note we fixed all random seeds in training for easy reproduction.

The models we have utilized are logistic regression, K nearest neighbor, classification tree, random forest, and support vector machine. Our most effective models turned out to be the SVM model, the Random Forest model, and the Logistic Regression model, respectively. Our goal here is to achieve the highest accuracy possible, as the class distribution is similar enough.

SVM:

We tried 4 kernel functions: *'linear', 'rbf', 'poly'* and *'sigmoid'* with varying penalty terms. SVM is able to create (with relative success) 20 different linear hyperplanes for the 20 different labels, and works relatively well when there is a clear margin of separation between classes. Additionally, SVM is effective in high dimension spaces and is relatively more memory efficient. The best support vector machine model achieved **42.86%** accuracy, using *'linear'* kernel and *C = 10*.

Decision Tree Classifier:

This is a greedy method, and the algorithm will choose the most optimal steps at each node of the tree to have the lowest impurity cost. Due to this, the decision tree classifier would overfit on the training set, and thus not perform as well when predicting on the testing set. On the other hand, because random forest takes the average across many trees, it ultimately reduces the variance of the model, and we see this with random forest implementation below. After optimizing parameters, we achieved accuracy of **30.56%** - not as high as random forest.

Random Forest:

Random forests are an ensemble method for classification. A forest of decision trees is constructed during training, in a greedy manner, as the models make the most optimal decision on each step of the tree. Random forest is invariant to feature scaling and translation. It also does automatic feature selection, creates non-linear decision boundaries without complicated feature engineering, and does not overfit as often as other nonlinear models. The biggest benefit to the Random Forest is that it is an ensemble method, and takes the average across thousands of individual trees, thus yielding an overall stronger result from a group of weaker models. The output of random forest model is the mode of the classes of the individual decision trees.

We tried models with different parameter *'n_estimators'* and *'max depth'*, and *'max_features'*. *'N_estimators'* adjusts represents the number of tree estimators created in the forest, so the number of votes. *'Max_depth'* limits height growth of each tree, and *'max_features'* is the size of the random subsets of features to consider when splitting a node. We also tried out feature ranking with recursive feature elimination and cross-validated selection of the best number of features, and found that accuracy decreased, so did not use this in our final parameters.

The best random forest model achieved **42.53%** accuracy, with parameter set *n_estimators = 100, max_depth = 10, max_features = 5*. With random forest, we can easily overfit, so we limit max depth, and the number of features considered per split.

K Nearest Neighbor:

KNN is a non-parametric method for classification. We can classify an image by a majority vote of its neighbors in a feature vector space, and the image will be assigned to the class which is most common among the image's k nearest neighbors. We tuned for different k values, measuring distance with euclidean distance, and found the best *k-value* as *10*, with **38.20%** accuracy. As k becomes larger, the accuracy of the training set decreased, which could indicate it is overfitting when k is small. This may be caused by lack of data.

KNN suffers from the curse of dimensionality; the classifier makes the base assumption that similar points share similar labels, aka nearby neighbors are similar to a point, but in high dimensional spaces, points that are drawn from a probability distribution tend to not be close together. Success when using the knn is very dependent on

having a dense data set, as knn requires a point to be close in every single dimension - it needs all points to be close along every axis in the data space

KNN works well with a small number of features, but struggles to predict as there are more features. As the number of dimensions increases the size of the data space increases, and the amount of data needed to maintain density also increases. KNN also requires to choose the optimal number of neighbors, it is sensitive to outliers as it chooses the neighbor based on the distance criteria, and requires an initial parameter of choosing the optimal number of neighbors.

Summary of Results:

SVM Model accuracy is: 42.8571 KNN Model accuracy is: 38.2060 Decision Tree Model accuracy is: 30.5648 RFM Model accuracy is: 42.5249 Logistic Model accuracy is: 41.5282

Table 4: *Accuracy values across all the models*

Discussion:

The most interesting features that our group noticed was **Fast_corner**, **SIFT**, and **BRIEF**. We believe this because when graphing the histograms across all the labels, we notice a clear variance among the different labels. **Fast_corner**, **SIFT** and **BRIEF**, all labels follows a similar distribution. For example, zebra, bear and penguins always return the highest number of keypoints, and comets, leopards and airplanes always return the lowest number of keypoints. This makes sense, because from our classification tree model, the highest accuracy received after using cross validation was found when utilizing all the features up to BRIEF.

Other features we found interesting are **harris** and **shi_tomasi**. This is because across all the models, the accuracy had the maximum increase when including the harris and shi_tomasi features. Additionally, from the Random Forest model, after cross validation, the highest accuracy was found for including features up to the Harris feature. The last useful feature we found was the **canny_edges**, which returns the sum of all the edge gradients. This is because including this feature improved the accuracy across all our models, and the SVM model had the highest accuracy for all the features up to the canny_edges feature.

As mentioned above, we thought ORB feature would be useful because it returns the number of keypoints. However, from figure 14, we see there is not a great amount of variation with the number of keypoints that the orb feature returns. Additionally, when going through cross validation and computing the accuracies for the successive features, we found the minimum increase in accuracy for the logistic regression model were from including the features prewiit_h and prewitt_v. We believe this makes sense because these two features returns the sum of a flattened matrix, and these sum values might not be too different across the different labels. From the classification tree and SVM models, it is found that the minimum increase in accuracy is found when the mean luminance and std_luminance features are included. Finally, from the random forest model, the minimum increase in accuracy is found when the ORB feature is added. From the histogram above, we see there are incremental differences in the number of keypoints returned across the labels.

Deep Neural Networks: GradProject_NB4.ipynb