W281 Final Project | Kevin Fu

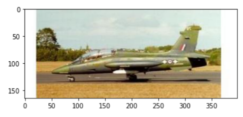
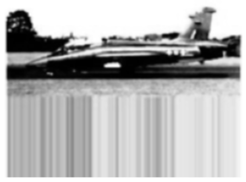
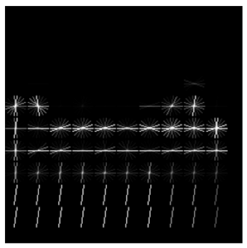
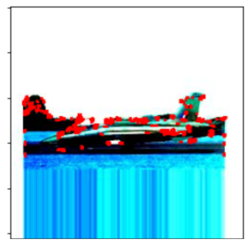
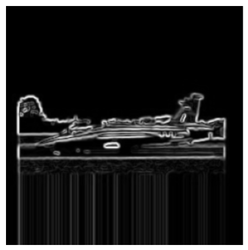
EDA

Excluding the .DS_Store file and .ipynb_checkpoints folder, there are a total of 2921 files within the "20_categories_training" folder, 1419 of which are corrupt jpg files. Subtracting a duplicate "crab" image ("crab_0001 (2).jpg"), we are left with a **total of 1501** valid images in the training dataset. We will use a 80/20 training/test split (1200 in the training set, 301 in the validation set).

We have a fairly evenly spread out dataset across all 20 image classes with 'gorilla', 'leopards', 'penguin' having the most training images. The average image size had a width of 403 and a height of 340. The largest image had a width of 3424 and a height of 3225; the smallest image had a width of 115 and a height of 105 (not necessarily the same image).

In terms of preprocessing, I first created a square padding of all images by padding the shorter side with the edge values until it is a square with the longest side of the image. We use edge values instead of 0's so that the mean and SD of the image is kept. The image is now a square with the aspect ratio intact. We can then resize and shrink our image down. Experts in our field suggest 224x224 for image classification[1], but since our average image size had a width of 403 and a height of 340, a nice compromise between the two is 256x256. Center cropping did not improve the explained variance or the classification models' accuracy, likely because the object of interest was not always located directly at the center in our training images. Lastly, we normalized the images using the mean and SD generated from the pretrained Imagenet.

Feature Extraction

Original	Preprocessed	Hog	Harris	Sobel
				

Before all of the following feature selection steps, I first converted the image to grayscale and used a Gaussian Blur so that edges are easier to detect.

Hog[2]: I implemented a Pseudo-Gridsearch for the best performing Hog parameters (see table below, bold column is the parameter set I ultimately used):

Pixels Per Cell	8x8	16x16	**24x24**	24x24	36x36
Cells Per Block	2x2	1x1	**1x1**	2x2	1x1
Explained Variance	40%	50%	**60%**	50%	70%

I used 9 orientations because the original research paper states, "increasing the number of orientation bins improves performance significantly up to about 9 bins, but makes little difference beyond this."[3] The pixels per cell has to be adjusted accordingly for the bias variance tradeoff (8x8 picks up more detail but more noise, 36x36 picks up less detail but less noise). I also noticed that the above jumps in explained variance does not necessarily implicate

[1] "Data Set Considerations." PowerAI Vision Version 1.1.3, https://www.ibm.com/docs/en/mvi/1.1.3?topic=sets-data-set-considerations.
[2] Waheed, Ahmed. "How to Apply Hog Feature Extraction in Python." Python Code, 30 Oct. 2020, https://www.thepythoncode.com/article/hog-feature-extraction-in-python.
[3] Dalal, Navneet, and Bill Triggs. Histograms of Oriented Gradients for Human Detection. https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf.

an increase in classification accuracy. 24x24 was a suitable compromise for high accuracy and appropriate bias variance tradeoff. I used a small cells-per-block parameter as a way to offset the large pixels-per-cell values above.

Harris[4, 5]: I implemented a Pseudo-Gridsearch for the best performing Harris Corners parameters (see table below, bold column is the parameter set I ultimately used):

Block Size	2	2	**4**
K Size	5	3	**3**
K	0.07	0.04	**0.04**
Explained Variance	20%	30%	**35%**

The Block Size denotes the size of the neighborhood of surrounding pixels considered. The ksize parameter denotes Sobel kernel size (large ksize: more pixels are part of each convolution process, resulting in blurrier edges). K is the Harris detector free parameter that adjusts the bias variance trade off (bigger k, less corners detected; smaller k, more corners detected). The Harris corners ended up being one of the worst performing feature sets in terms of classification accuracy with 20 or less features.

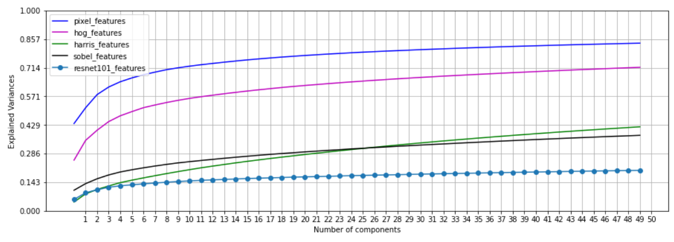
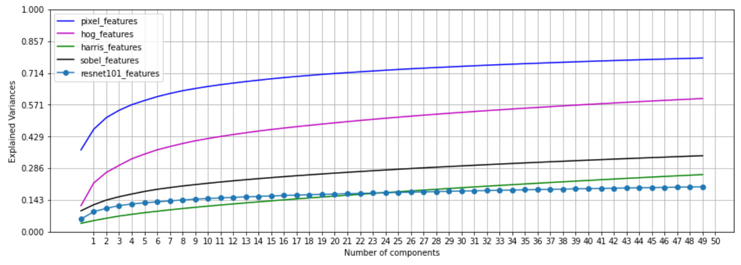
Sobel[6, 7]: I chose the Skimage Sobel edge detector that combines both the x and y derivatives to capture gradients on both axes (dx=1, dy=1). Using only 1 of the 2 axes captures half of the explained variance vs. using both axes (15% vs. 32%). I experimented with a ddepth parameter of CV_64F so that more information from the original image would be kept[8]. I also tried using a variety of Sobel kernel sizes (1, 3, 5, 7) to convolute with the original image. However, these parameters did not perform as well as the default parameters in the Skimage Sobel edge detector.

Pixel: No special parameters here. I did not use the Gaussian Blur because we are performing pure pixel analysis. The most interesting finding was that there was a 3% increase in explained variance when the images were not normalized (86.5% vs. 83% normalized images).

Resnet: Using Google Colab Pro's GPU, each run to extract features from the pretrained Resnet model took 5hrs 45min. Unlike the above, we are unable to visualize the features extracted. Lastly, even though we are using a complex pretrained model to extract features, the Resnet features did not outperform the other models when it comes to explained variance or accuracy.

<u>Feature Visualization</u>

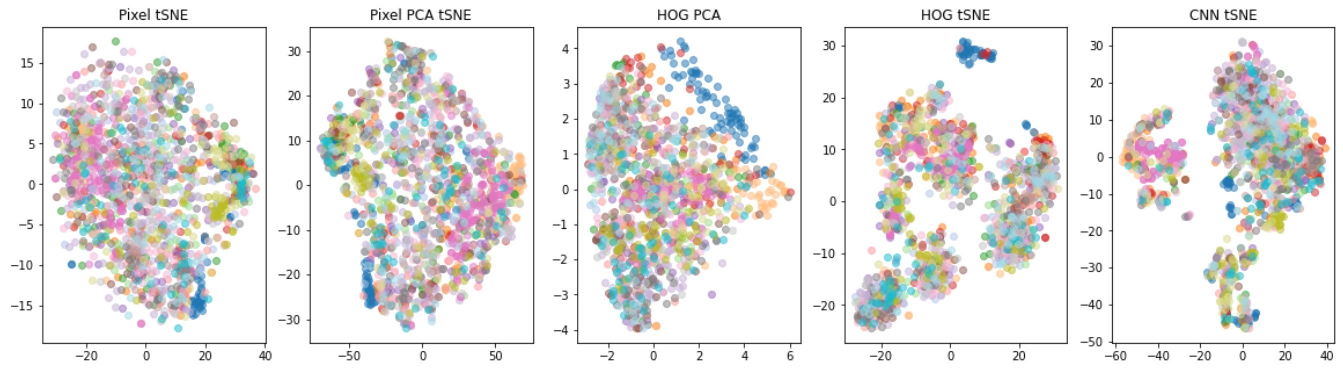
All features except for Resnet and Sobel explained more variance with less features **without** normalization. Despite this, due to standard practice, I moved forward with using the normalized features in the final analysis.

Explained Var Curves: No Normalization	Explained Var Curves: Normalization
	

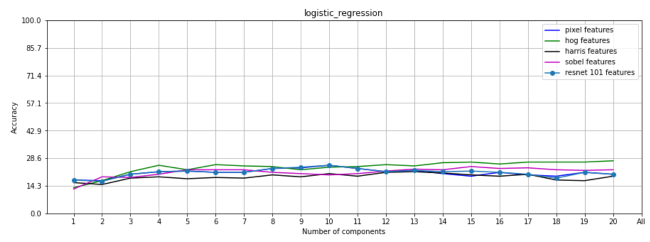
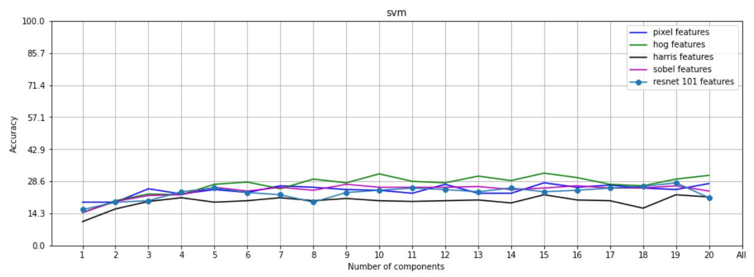
Blue[9] (Airplanes) and Pink (Leopards) have very obvious clusters - especially noticeable in the Pixel PCA tSNE plot and the Hog tSNE plot. The other clusters are not as distinguishable as we would like them to be. Because the tSNE

[4] Botforge. "Harris Corner Detection." BOTFORGE, 2 Oct. 2016, https://botforge.wordpress.com/2016/10/02/harris-corner-detection/.
[5] "What Does Ksize and K Mean in Cornerharris?" *Stack Overflow*, https://stackoverflow.com/questions/54720646/what-does-ksize-and-k-mean-in-cornerharris.
[6] "Python Program to Detect the Edges of an Image Using Opencv: Sobel Edge Detection Method." GeeksforGeeks, https://www.geeksforgeeks.org/python-program-to-detect-the-edges-of-an-image-using-opencv-sobel-edge-detection/.
[7] "Edge Detection Using Opencv." LearnOpenCV, 15 July 2021, https://learnopencv.com/edge-detection-using-opencv/.
[8] "Image Gradients." https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_gradients/py_gradients.html.
[9] "Choosing Colormaps in Matplotlib" Choosing Colormaps in Matplotlib - Matplotlib 3.5.1 Documentation, https://matplotlib.org/stable/tutorials/colors/colormaps.html.

tends to do better than the PCA plots, we will assume our data requires a nonlinear transformation and use tSNE. Again, this clustering is more obvious in the non-normalized version of these plots (see Github).



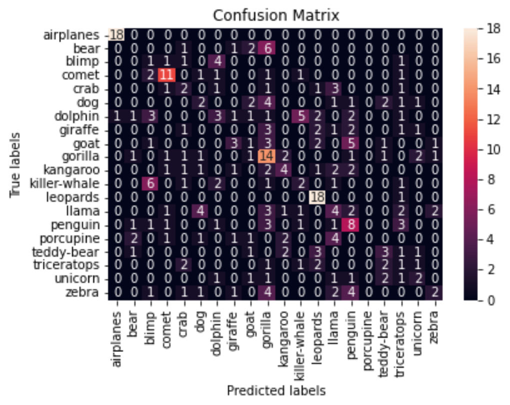
Models & Accuracy Curves (See notebook for more plots)

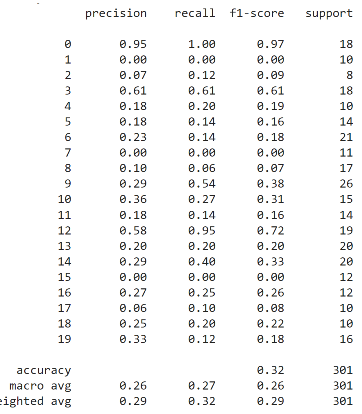
Logistic Regression (Normalized Features)	SVM (Normalized Features)
	

Among the normalized feature set, SVM was the best outperformer of the three (25-30% acc), Logistic Regression second (20-30% acc), and the KNN model last (15-25% acc). Most models performed slightly better with non-normalized features. Gorilla was the most commonly predicted class by all models, regardless of accuracy.

Results (See notebook for more plots)

Hog SVM was the best performer (31.9% F1 Score). This result was very frustrating since I had achieved a 46.3% accuracy with a standalone CNN model when I initially started this project. This 46.3% was achieved with no preprocessing, no feature selection, and a simple resizing (300, 300) that did not preserve the original aspect ratio.




```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        18
           1       0.00      0.00      0.00        10
           2       0.07      0.12      0.09         8
           3       0.61      0.61      0.61        18
           4       0.18      0.20      0.19        10
           5       0.18      0.14      0.16        14
           6       0.23      0.14      0.18        21
           7       0.00      0.00      0.00        11
           8       0.10      0.06      0.07        17
           9       0.29      0.54      0.38        26
          10       0.36      0.27      0.31        15
          11       0.18      0.14      0.16        14
          12       0.58      0.95      0.72        19
          13       0.20      0.20      0.20        20
          14       0.29      0.40      0.33        20
          15       0.00      0.00      0.00        12
          16       0.27      0.25      0.26        12
          17       0.06      0.10      0.08        10
          18       0.25      0.20      0.22        10
          19       0.33      0.12      0.18        16

    accuracy                           0.32       301
   macro avg       0.26      0.27      0.26       301
weighted avg       0.29      0.32      0.29       301
```

Airplanes (97% F1 score) and Leopards (72% F1 score) were the easiest for the classifier to recognize. Bears, Giraffes, and Porcupines (0% F1 score) were among the most difficult for the classifier to recognize. Blimps vs. Killer Whales and Bears vs. Gorillas were among the most commonly confused pairs. These mistakes were not completely off-base as these objects do have a similar shape. With more training images, the model could likely do better.