Hand-Drawn Sketch Recognition

Jeremy Lan
Michelle Lee
Tae Kim
Steven Sung

I. Abstract

Approaches to solve multiclass image classification problems have been well-documented, but exploring the classification of human-drawn sketches requires further exploration. In this project, we attempt to characterize the Sketchy dataset and explore options for accurate and efficient multiclass classification.

We examined many feature engineering approaches and evaluated their effectiveness in the context of our dataset, where our sketch images were often low-detail and lacking the depth information that many image-based feature extraction techniques rely on. Despite some limitations, we found that HOG and SIFT descriptors still provided classification value. We also explored dimensionality reduction via PCA and optimized our feature engineering parameters to maximize their effectiveness.

Random forest, support vector machine, and convolutional neural network models were trained on our data and our generated features. In preliminary results, we found that using pre-trained CNN architectures like ResNet50 and VGG16 yielded superior results to the RF and SVM models trained on image features. We were then able to adjust our CNN models to intake the HOG vector in addition to the base image, which improved performance significantly. This novel approach allowed us to reach a test accuracy of 91.4% which far exceeds the objective baseline accuracy of 0.8%.

II. Introduction

Multiclass image classification has been a central research area in computer vision for many years. Although some real-life applications can be addressed with binary classification methods, such as determining whether a medical tissue image or skin lesion is cancerous, recognizing everyday objects in the world is essentially a multiclass classification problem. For instance, autonomous vehicles must correctly identify objects in images, such as vehicles, buildings, and pedestrians, so that the vehicle's software can make informed decisions about the surrounding environment.

Although digital cameras are capable of faithfully representing a scene in the world, the same can not be said regarding human sketches. Often, two individuals will produce wildly different sketches, even when they are attempting to recreate the same subject from a photo or in real

life. This variability is due to the inherent subjectivity and limitations of human perception and drawing ability, which can result in distorted features, simplified forms, and anthropomorphized objects. Nevertheless, sketches are usually easily recognizable by other humans, so the task of developing models to master sketch-based image retrieval is a unique challenge that has not been as thoroughly explored as traditional image classification.

In this report, we explore the distinct challenge of sketch classification by exploring various features and models. We engineer both simple and complex models and evaluate a range of machine learning algorithms to develop an accurate classification model for the provided dataset.

III. Dataset

The Sketchy database is the first large-scale collection of sketch-photo pairs. The data is founded on 12,500 photos of objects (100 per category, 125 categories) - these photos were then used to crowdsource sketches prompted by these photos. In total, 75,471 sketches are provided, with around 600 sketches per category and post processing applied to the raw sketch to center and uniformly scale each image. Each sketch is mapped to an original object photo by the file name, as shown below.

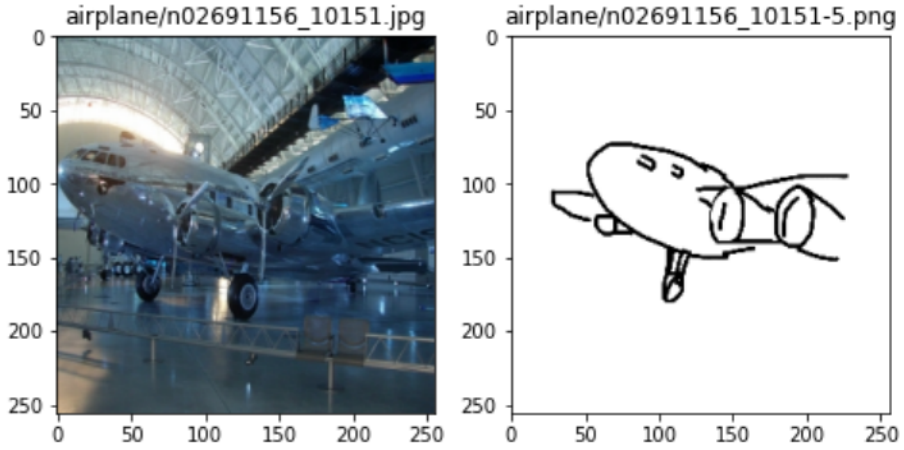

Figure 1: Example Image-Sketch Pair

The 125 image categories were selected by the database creators to meet the following criteria: exhaustive, recognizable, and specific. Thus, sketch categories cover a large number of common objects with recognizable sketch representations. All sketches contain grayscale data only, with black sketch marks and a white background.

Dataset Challenge: Varying Photo Interpretation

For each photo-to-sketch combination, we found that there are wide variations between them. One of the most noticeable differences is the caricature-like features of the sketches. For example, the wings of the bee below are larger than what is portrayed in the photo. Different

sized features becomes a challenge when training a model since the features may not be consistent and requires a feature extraction that adjusts the parameters with size in mind.

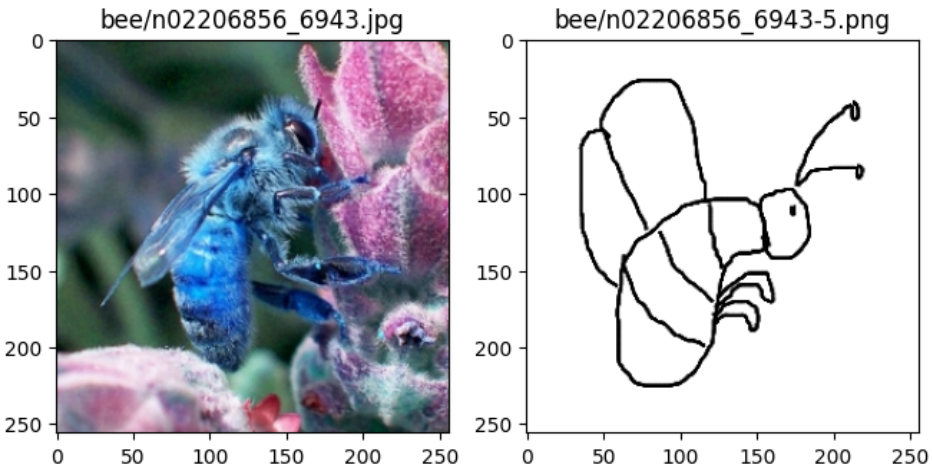

Figure 2: Example of Human-induced Caricature for image features

There are also varying details of the same photo. Individuals may perceive images differently where features of the photo are either obscured or removed overall. The sign in the center of the bell for example is gone for one of the sketches, making feature extraction more difficult since unique features of sketches from the same category can vastly differ.

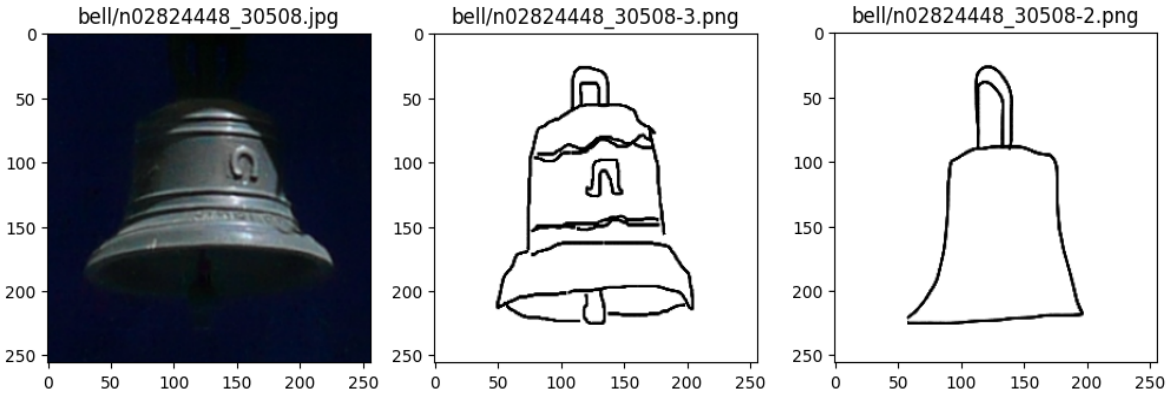

Figure 3: Example of variability of sketch and feature detail

The large number of categories makes defining and predicting different objects adds additional challenges. With a baseline model accuracy of 0.8% (125 balanced categories), we'd require advanced features to perform better than randomly predicting sketches. Additionally, some categories such as cats, dogs, and other four-legged creatures can easily be mistaken for each other, and it will be interesting to see which categories were the best (and worst) classified.

IV. Features

Feature Engineering and Extraction

Despite the apparent lack of detail in sketches when compared to standard images, there are still many valuable features beyond the original sketch data that we are able to extract.

Canny Edge Detector:
To reduce detail variability, we attempted to use canny edge detectors since it is a simple feature that reduces the feature size. The detector works on simpler images with less details such as apples really well. However, with more complex objects like airplanes, it creates more noise. In both cases, there are double borders as a consequence due to the double gradient calculations. The edge detector assumes the black border as a surface that's shaded in. Therefore, it calculates a gradient from the white background to the black border and then from the black border towards the white object. However, this took only two minutes to apply to all of the images, so preprocessing is simpler.

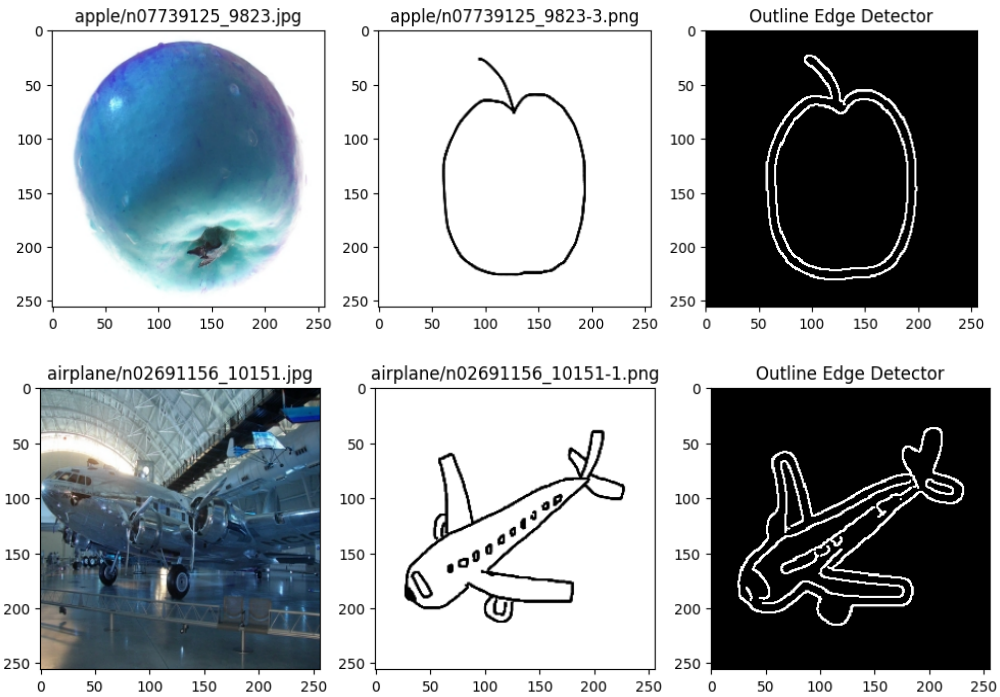

Figure 4: Example of Canny-Edge detector feature implementation

Histogram of Gradients (HoG):
The HoG feature is used to capture the texture and shape of an object in an image. By calculating the histogram of gradient orientations and magnitudes across an image, HoG descriptors provide an object representation that is robust to changes in scale and orientation.

We calculated HoG descriptors for each image through the use of the *hog* package within the *skimage.feature* library. In the below examples, object features like the motorcycle wheels and snail shell are captured effectively within the HoG descriptor which should assist in object classification when implemented in our models.

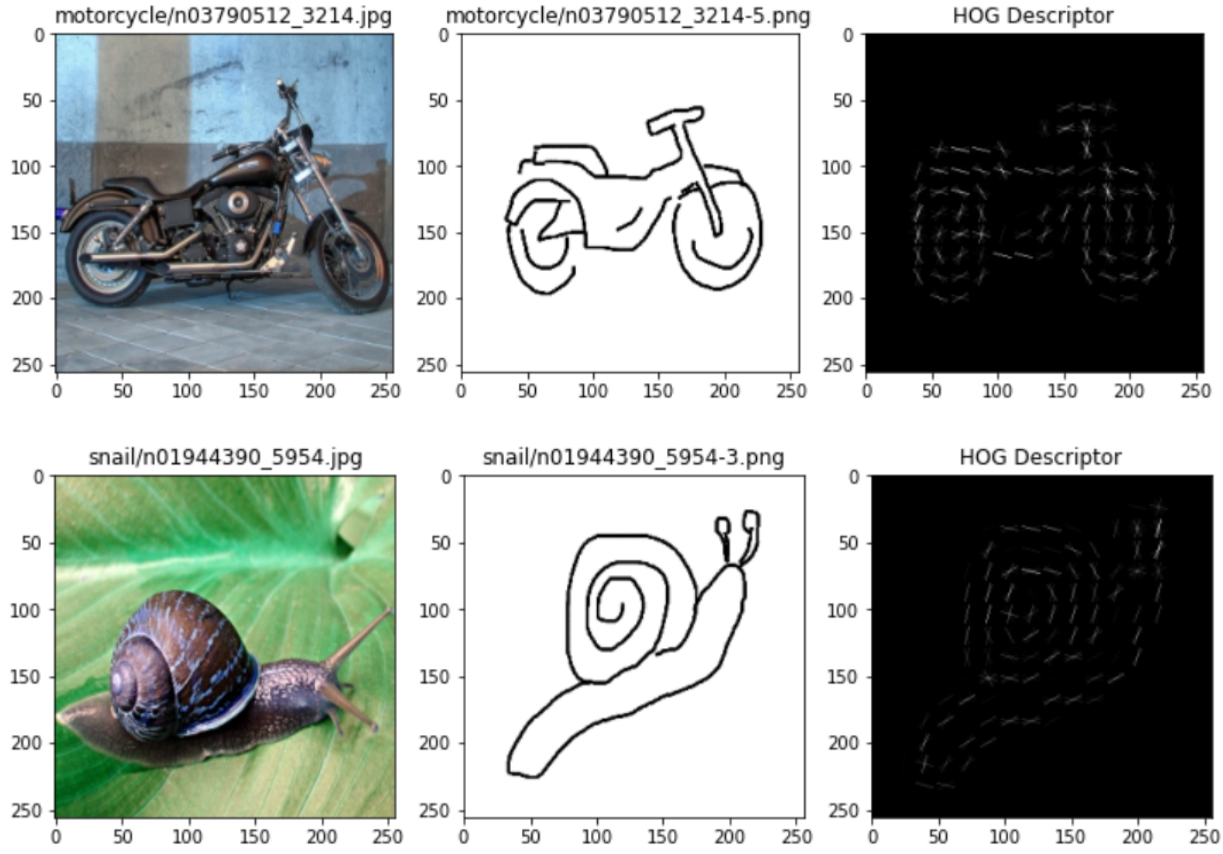

Figure 5: Example of Histogram of Gradient feature implementation

Local Binary Pattern (LBP)**:**

The LBP descriptor captures local texture information of an image by comparing pixels in a local area to the central pixel, and historically has been used in facial and motion analysis. However, given the “simplified” nature of our sketches with no background information and little shading, we did not find LBP as useful. From the sample image below, we can see that not much additional value is generated from the LBP feature - we decided not to pursue this feature in our model design.

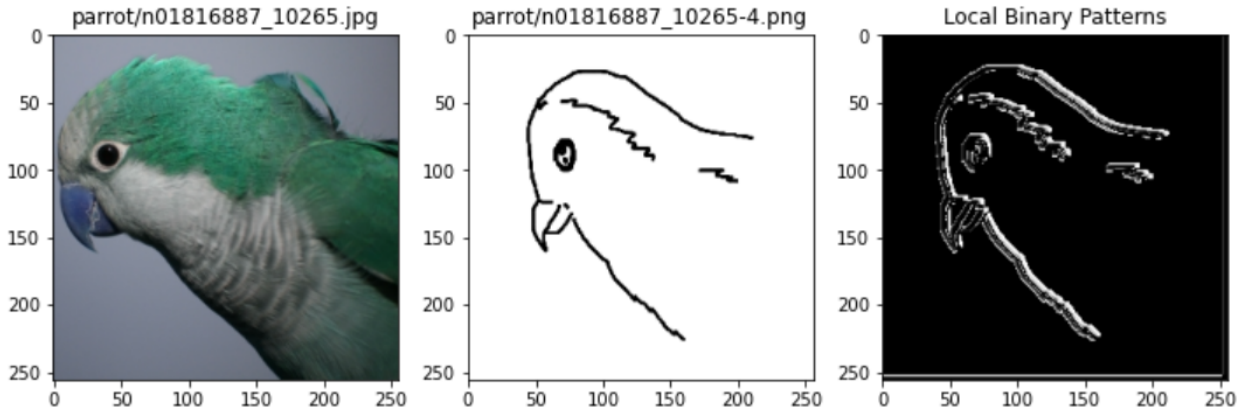

Figure 6: Example of Local Binary Pattern feature implementation

Scale-Invariant Feature Transform (SIFT)**:**

The Scale-Invariant Feature Transform (SIFT) is a feature extraction technique that extracts distinctive keypoints from an image that can be used in later identification, even if the image is scaled or rotated. Using the SIFT_create function within cv2, we were able to generate SIFT features for each image with manual tuning for the maximum number of keypoints, contrast threshold, and filter deviation in the keypoint detection process. We found SIFT useful for extracting features capturing distinctive shapes, even with the lack of details in the sketch data.

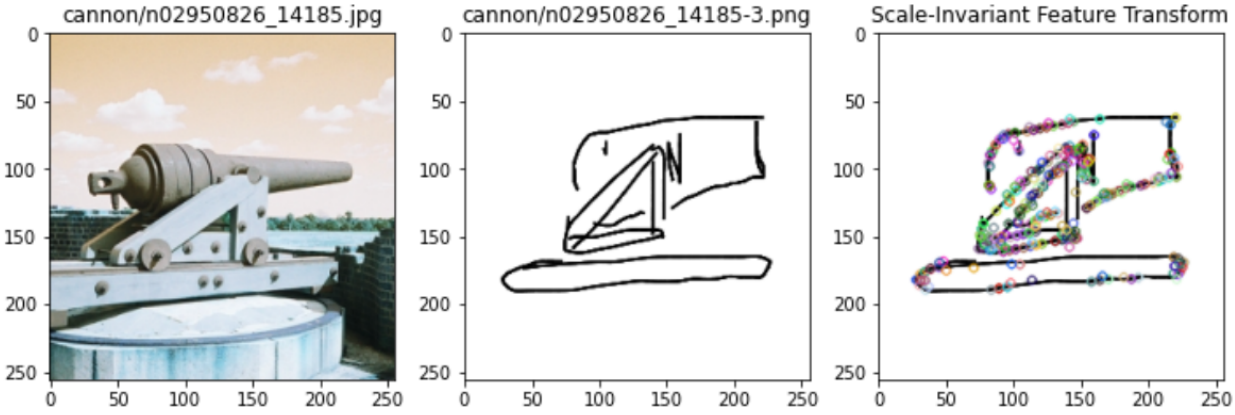

Figure 7: Example of SIFT feature implementation

PCA Dimensionality Reduction**:**

Since each image is made up of 256x256 pixels, this is inherently a 65536-dimensional classification problem. Despite the lack of detail in our sketches, we desired to use PCA to see if there could be opportunities to increase model training efficiency via dimensionality reduction, while still preserving important information contained in sketches. Below, we identified the top 20 principal components and projected the input sketches onto a linear combination of these top dimensions.

```python
# Plot Top 20 basis representations
for k in range(20):
    e = D2@vec[:,k]
    plt.subplot(4,5,k+1)
    plt.imshow(e.reshape(xdim, ydim), cmap = 'gray')
    plt.axis('off')
```

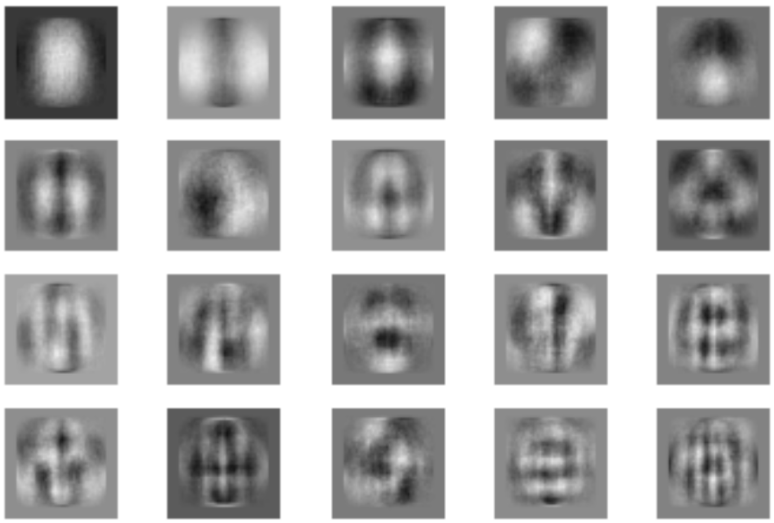

Figure 8: Visualization of top 20 principal components

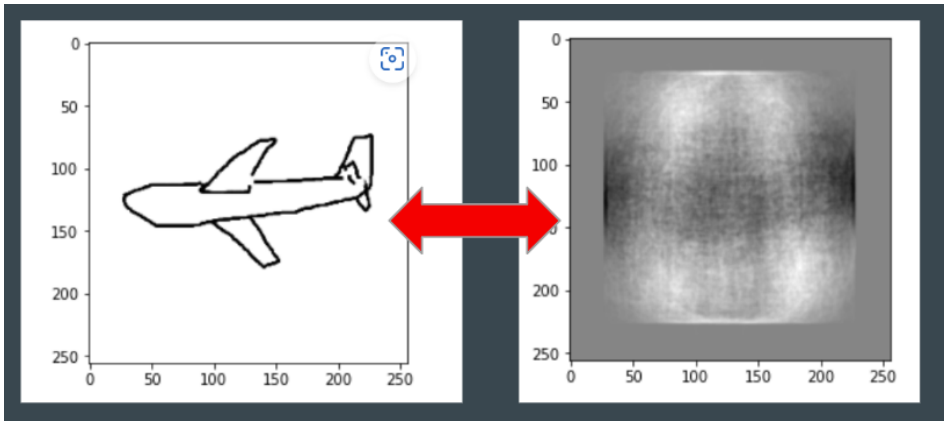

Figure 9: Reconstruction of original sketch via top 20 principal components

Although it may appear that most of the image detail is lost when projected onto this new basis, we were able to use these reduced-dimensional representations to identify the nearest neighbors between two sketches. While we anticipate a loss in model performance when dimensionality reduction is applied, we are interested to see what improvements are made with regards to model efficiency.

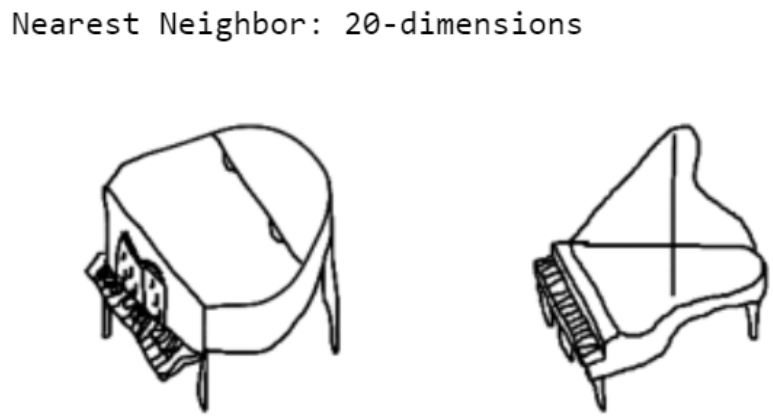

Figure 10: Sketch “nearest neighbor” identified via PCA decomposition

V. Model Development

Random Forest (RF)

Random forests are formed by combining decision trees, where each tree is trained on a random subset of data and features. In general, RF models are robust to handle noise and mitigate overfitting from any individual tree. GridSearch can also be performed to optimize RF parameters, including the number of estimators, max tree depth, and minimum samples required for node-splitting.

Support Vector Machines (SVM)

The support vector machine algorithm works in classification applications by finding an optimal hyperplane that best maximizes the margin between the hyperplane and the nearest data points from each class. For multiclass classification, classifiers are often trained in a way that directly classifies instances into multiple classes, known as one-vs-all (OVA) classification - this is also the default mode for scikit-learn’s SVC function. Other function parameters we evaluated were the type of kernel (linear) and the penalty parameter (balanced for generalization and accuracy).

Convolutional Neural Networks (CNN)

Deep convolutional neural networks with many hidden layers typically excel in image classification tasks. For this project, we used two well-known CNN architectures used often in literature. ResNet50 is a 50-layer CNN trained on approximately one million images from ImageNet with roughly 23 million parameters. VGG16 is a 16-layer CNN also trained on the same image dataset from ImageNet. We constructed two tensorflow models based on these two architectures by downloading their pre-trained weights and prepending an input layer to take in our sketched images of size (256x256x3). Both models were then appended with two fully connected layers of size 1024 and 512, followed by a final output layer with 125 nodes with a

softmax activation function to construct a multiclass categorization model. Both were trained with 10 epochs with an early stopping mechanism to stop training if the validation loss does not improve in 3 epochs to prevent overfitting.

We then constructed another set of tensorflow models capable of taking in an additional feature vector as input. Of all our extracted features, HoG showed the most promise in our preliminary study and was thus chosen to be integrated into our model. We used the functional tensorflow model API to construct a model with two inputs: an image of size (256x256x3) and a HoG vector of size (2048x1). The image input was fed into the ResNet50 layers, and its output was concatenated with the other input vector. This combined vector was then fed into a final output layer with 125 nodes with softmax activation to achieve a multiclass categorization model.

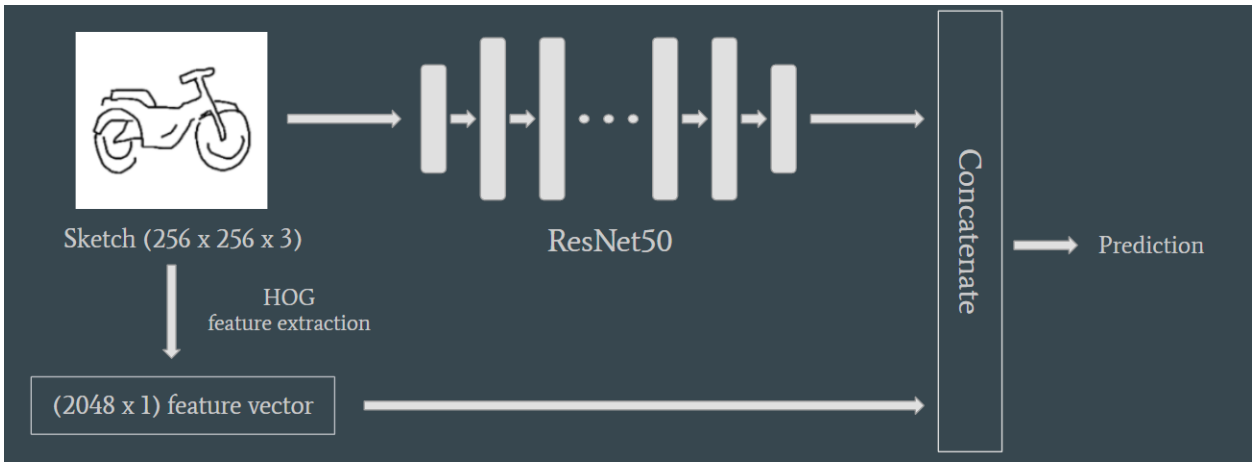

Figure 11: Structure of CNN model with an additional feature vector as input

VI. Model Performance and Results

Random Forest

A random forest classifier was trained on our generated HOG features - mostly due to the ability of random forest models to handle high-dimensional feature spaces. GridSearch was implemented to optimize the model via hyperparameter tuning, which led to a final model accuracy of 43.4%. The training time for each model varied between 3 and 15 minutes, depending on model complexity.

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
}
```

Support Vector Machines

SVM models were parameterized and run on our engineered SIFT and HOG features, as well as on our reduced-dimensionality image representations as generated through PCA.

For our HOG-SVM model, performance was evaluated on a linear kernel, with a final accuracy of 46.1%, which slightly outperforms the Random Forest model run on the same feature.

When running an SVM model on our SIFT features, we ran into several constraints which limited effectiveness. We first encountered memory issues on the full dataset and thus ended up
SIFT - constraints on memory error, thus only 10% of the data was used to train (50 images per category). Since each image had a different number of SIFT features identified, the SIFT vector lengths across images was inconsistent and padding was required to standardize feature shape.

Due to the complex nature of our SIFT feature, performance was evaluated on both a Linear and Radial Basis (RBF) kernels. Linear kernels are faster and often more memory-efficient, but RBF kernels perform better when the data is more complex and not linearly separable. Model accuracy for both linear and RBF kernels was low at approximately 8%.

To improve SVM performance on our SIFT features, k-means clustering was performed on the descriptors to identify cluster centers and assign each image descriptor to its nearest cluster center. The new “bag-of-words” feature vector is generated for each image by counting the frequency of each cluster center in the image. Although adding the feature added an additional 23 minutes, running SVM on these bag-of-words features improved performance, as shown in Table 1 below.

Kernel Type	KMeans?	Accuracy	Training Time
Linear	No	8.4%	12 mins
RBF	No	7.8%	14 mins
Linear	Yes	13.7%	5 seconds
RBF	Yes	15.8%	5 seconds

Table 1: Results of SIFT-SVM model with varying parameter tuning

We converted our images to lower-dimension bases using PCA and trained SVM models on these reduced representations. PCA was performed twice, to reduce the dataset to 9 and then 20 dimensions. Classification accuracy improved to 27.2% with the 20-dimension representation, however with the tradeoff of increased training time.

The low accuracy of the PCA-reduced data also draws into question the usefulness of dimensionality reduction for our dataset specifically. Since our sketch data is relatively low-detail with a consistent white background and black contour that all images share, PCA is not as

effective in preserving the detailed characteristics of individual classes in a basis reduction. Compared to more popular literature examples like the “eigenfaces” study, it is thus much more difficult to identify common features across classes, which can also explain the obscure bases generated as seen in figures 8 and 9. Despite these drawbacks, we still do see performance improvements compared to other features as well as compared to our model baselines, indicating that some value is still being extracted from this basis reduction.

PCA Dimension	Accuracy	Training Time
9-dim	20.5%	3 hours
20-dim	27.2%	10 hours

Table 2: Results of SIFT model on PCA-reduced data of varying dimension

Convolutional Neural Networks

Of our two initial CNN models relying solely on the sketch images, the ResNet50-based model marginally outperformed the VGG16-based model. Both models began to overfit the training dataset before the end of the 10 epochs, with the VGG16 model triggering our early stopping mechanism in just 4 epochs as opposed to 7 with the ResNet50 architecture. Both models completed training in a reasonable 30 minutes on a GPU-enabled machine.

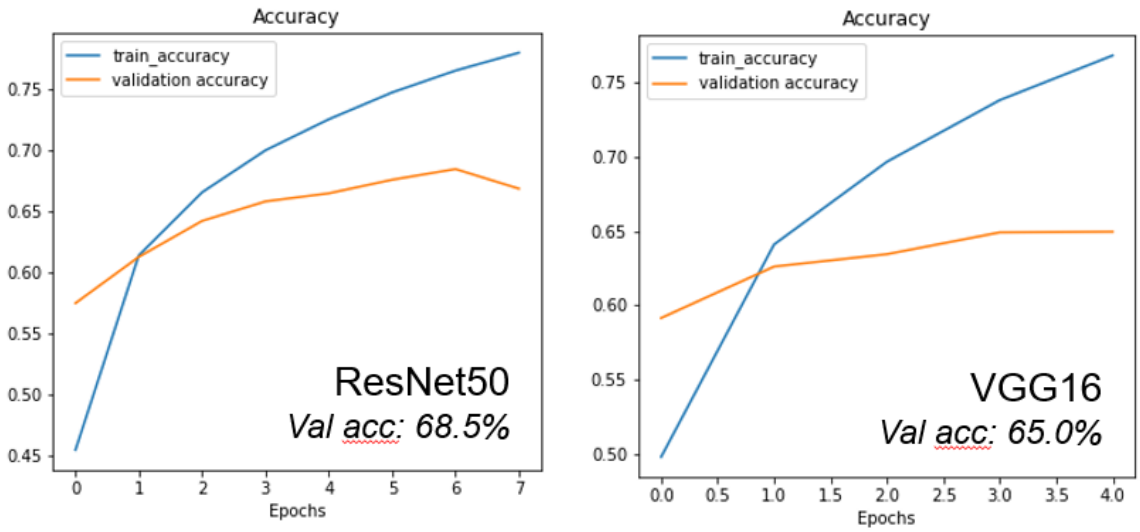

Figure 12: ResNet50 and VGG16 models performance without the addition of extracted features

When supplied with the extracted HoG descriptor feature vector as additional input, the model performed significantly better, improving validation accuracy from 68.5% to 91.5% with no signs of overfitting the training set for all 10 epochs. This suggests that the model performance could be further improved by supplying a larger dataset.

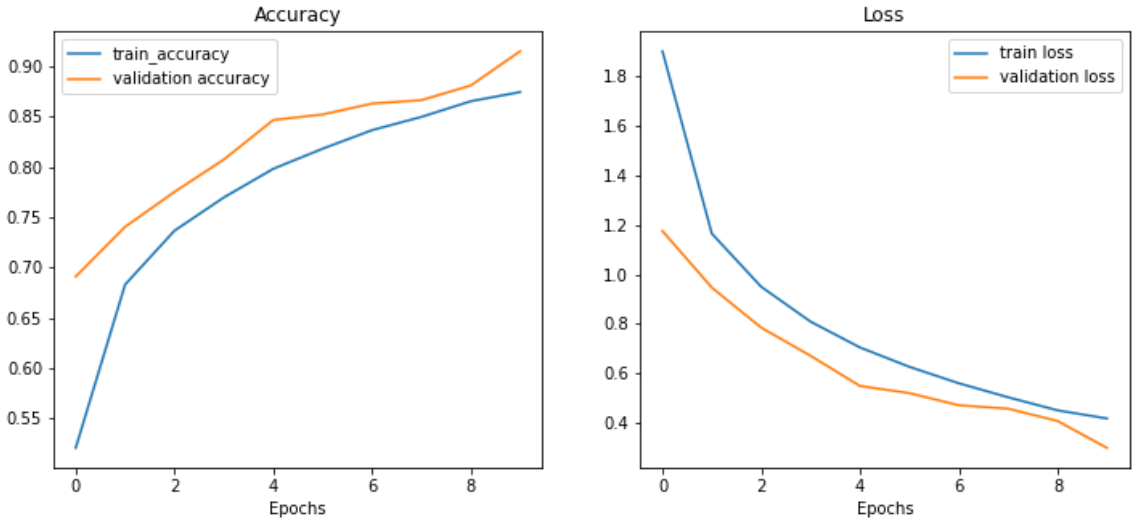

Figure 13: Performance of CNN model with HoG feature vector as additional input

Summarized Model Performances

Model	Features	Accuracy	Training Time
SVM	SIFT	5.1%	38m
SVM (10% data)	SIFT-KMeans-Rbf	15.8%	5s
SVM	PCA Image	27.2%	10h
Random Forest	HOG	43.4%	15m
SVM	HOG	46.1%	1h 35m
VGG16 CNN	Image	65.0%	30m
ResNet50 CNN	Image	68.5%	30m
ResNet50 CNN	Image + HOG	91.5%	2h

Table 3: Performance of all model types with their corresponding validation accuracy and total training time

Despite the seemingly low accuracy of some of our models, it is important to note the complexity of the classification task at-hand, with 0.8% baseline accuracy. Additionally, it is referenced that human recognition accuracy was set at around 73% (asking humans to classify images). Thus, even models with accuracies of 40% significantly are outperforming the objective baseline, and our neural network models show improved performance even over a human baseline once additional features are integrated.

VII. Discussion

Generalizability

The image dataset was randomly split into train, validation, and test sets with a 80:10:10 ratio. The validation dataset was used to evaluate which combination of model architecture, hyperparameters, and extracted features performed the best, while the test dataset was used to evaluate the performance of our final model.

With regards to model tuning, we found GridSearch useful on our random forest models to optimize model performance. For our SVM models, both linear and radial basis kernels were explored. We also focused our parameter tuning efforts at the feature engineering level, by adjusting function parameters to generate effective HOG and SIFT features that minimized unwanted noise. Our exploration into PCA and dimensionality reduction also illustrated the tradeoff between runtime and accuracy that occurs as our data is compressed onto the lower-basis representations.

Our best model, a ResNet50-based CNN concatenated to take in an additional HoG descriptor feature vector, achieved 91.4% test accuracy (vs. 91.5% validation accuracy). The best performing categories tended to have simple, unique features that can easily be identified, such as the large rotors of a helicopter. They were often simple and easy to draw in a short amount of time. The sketches were also not very varied in terms of their perspective, angle, or part of the object that was sketched.

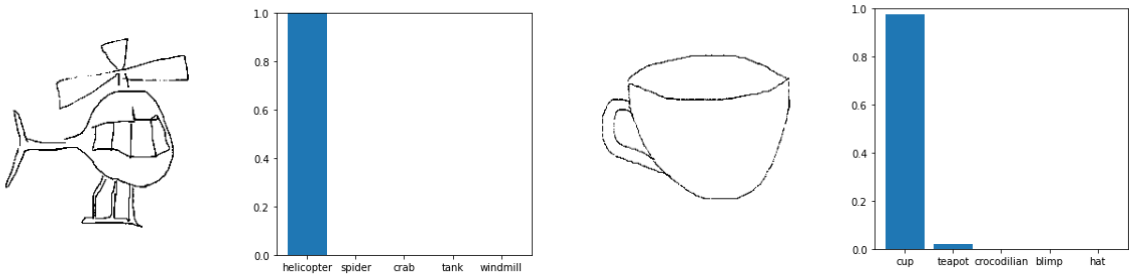

Figure 14: Best performing categories

The worst performing categories tended to have features that could be confused for other labels. For example, *Figure 15* shows that the model confused the large ears of the pig as wings for a bat, although pig was its close second guess. The sketches also had inconsistent features and often were complex concepts that were hard to draw in a short period of time.

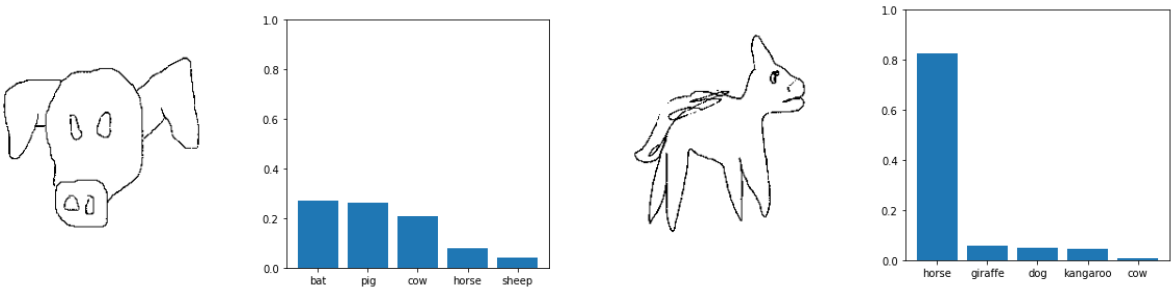

Figure 15: Worst performing categories

The model was repackaged to be able to perform on-the-fly predictions on any user inputted sketch to prepare for a live demonstration where a user would draw a sketch given one of the 125 classes the model was trained on. The model, however, did not perform as well as expected. We believe this is because of the nuance in which the dataset was collected. The dataset is *not* a collection of images where users drew whatever came into mind given the category, but instead drew the sketches based on specific photographs. Therefore, it might not generalize well to a live demonstration. In the future, if we want to accommodate such live demonstrations, we may want to re-train the model with a better fitted dataset.

Efficiency vs Accuracy

All CNN training was performed on an NVIDIA RTX 2060 with 12GB GDDR6 video memory and 32GB of RAM. With the CUDA developer's kit installed, tensorflow was able to complete training much faster than on an equivalent machine without a GPU. The two CNN models based on ResNet50 and VGG16 that only took in images as input both completed training in roughly 30 minutes. The models that were provided additional HoG descriptor feature vectors as input, however, took roughly 2 hours to complete training. We believe this is not due to the additional compute required by the feature vector itself, but due to the lack of optimization. Although tensorflow's built-in image dataset loader shuffles and batches images for you, we could not take advantage of this when used in conjunction with the additional feature vector as we had to ensure the vector was ordered in the exact same way. Therefore, we loaded in both the images and the feature vectors without any batching or shuffling, zipped them together as a single tensorflow dataset, *then* performed shuffling and batching. In the future, I would like to examine further how exactly tensorflow's built-in image dataset generator is optimized to apply the same techniques in our augmented dataset.

While it was clear that utilizing pre-trained CNN architectures yielded the most accurate results, these types of models are extremely complex and can often be very energy-intensive to train (especially without access to powerful computing resources). Consequently, there is still value in optimizing models for training efficiency over accuracy. When training models on the HOG descriptor for each image, we found that our tuned random forest model was more efficient than the comparable CNN and SVM models, with a minor dropoff in accuracy compared to the SVM model.

We also found that more aggressive dimensionality reduction yielded improvements in model training efficiency, but with the tradeoff of decreased performance. This can be seen in our PCA-SVM models. Although yielding undesirable results, PCA and similar techniques may become necessary depending on the quantity of training data.

Even when comparing CNN architectures trained on the same computing power, it was shown that augmenting the CNN input with the HOG feature quadrupled the training time (Table 3). The subsequent accuracy improvement is hard to discount, but there could be applications where a simpler CNN that's tuned for efficiency could be more desirable.

References and Links

Sangkloy, P., Lu, J., Fang, C., Yu, F., & Hays, J. (2016). The Sketchy Database: Learning to Retrieve Badly Drawn Bunnies. ACM Transactions on Graphics (TOG), 35(4), 1-12.

taekim027/2023-mids-w281-final-project (github.com)