Final Project Report

INFO 290T: Computer Vision

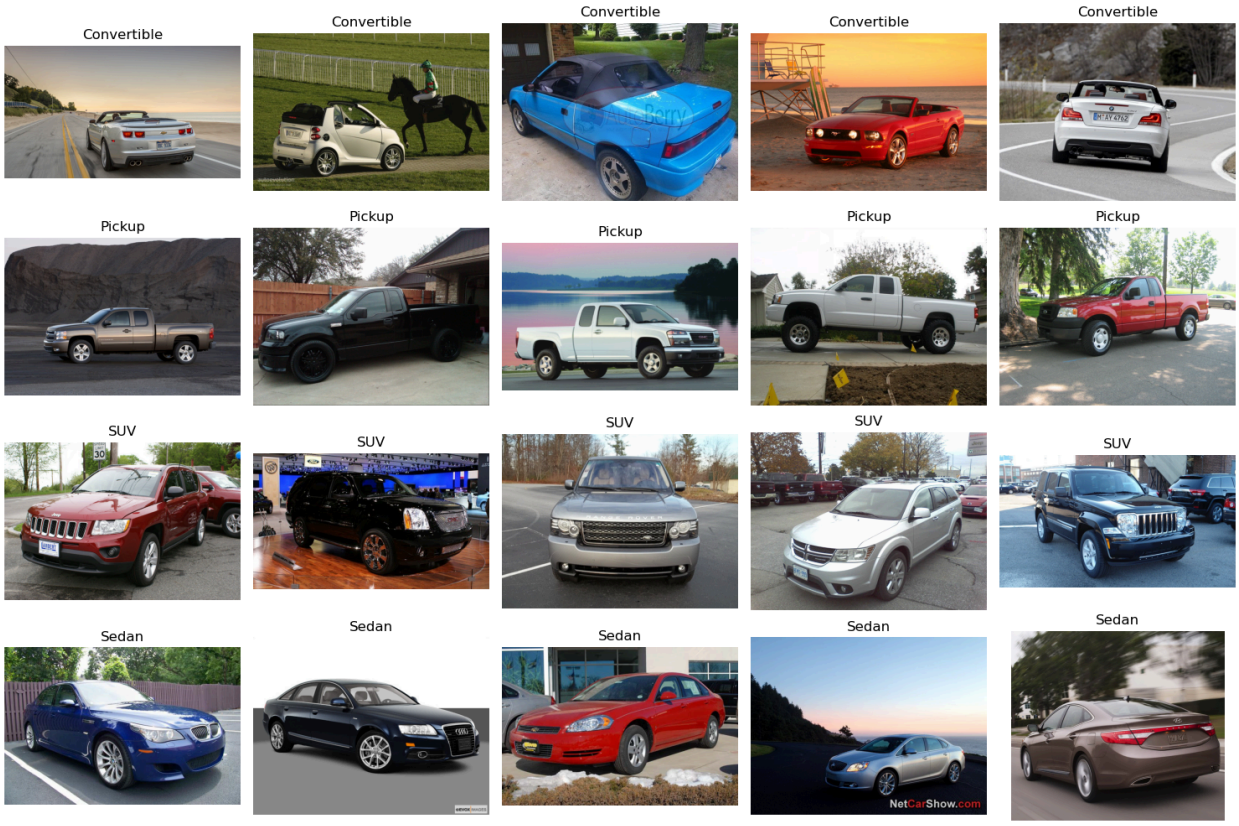
Stanford Cars Dataset

Team Members: Isaac Liu, Mayank Sethi, Gaurav Sharan Srivastava, Ashutosh Tiwari

Dataset Link: Stanford Cars Dataset [1], [2]

Code: Github

Sample Images:



Final Project Report

INFO 290T: Computer Vision

Overview

This project focuses on the nuanced recognition and understanding of objects from multiple perspectives. This complexity arises from the need to accurately identify and classify objects, in this case, cars, based on subtle differences in their appearances. For example, a sedan might look like a SUV from the front side. The project leverages a comprehensive dataset from Stanford University, which is meticulously curated to include 16,185 JPEG images spanning 196 classes of cars.

Classification Problem

The primary aim of this project is to develop robust models capable of recognizing and differentiating between various car types with high accuracy. This involves tackling the inherent complexity of 3D object representations, where the objective is to understand and identify cars from different angles and under varying lighting conditions. We seek to ensure high-quality data by excluding low-resolution images and correcting mislabeled entries, thereby improving model performance in distinguishing between nuanced vehicle categories.

Categories

We have selected items from some of the 196 original classes into four broad categories based on the car body style: Convertible, Pickup, SUV, and Sedan. The categories were constructed from portions of the provided class labels when they contained relevant information ("Acura TL Sedan 2012" became Sedan), with further online research used in ambiguous cases ("Ford F-150 Regular Cab 2012" became Pickup).

Size/Resolution

The original dataset contained a very wide variety of image resolutions. Upon visual inspection, we determined images smaller than 350 by 200 we of too poor quality to use in our classifier. 256 by 256 is a standard resolution for computer vision tasks, and we selected that as the size of our final processed data. We did not do any upsampling, and thus removed images smaller than this, making our overall restriction 350 by 256. Here is a description of the original resolutions of the images that fit these criteria:

	Count	Mean	Standard Deviation	Minimum	25th Percentile	50th Percentile	75th Percentile	Maximum
Width	7,492.00	812.10	447.70	358.00	584.00	640.00	1,024.00	7,800.00
Height	7,492.00	564.48	314.89	256.00	375.00	480.00	680.00	5,400.00
Number of Pixels	7,492.00	595,078.80	1,100,151.84	95,944.00	219,636.00	307,200.00	691,200.00	42,120,000.00

To resize our images, we first downsampled all images (with linear interpolation) with both dimensions larger than 260 by a scaling factor such that the smallest dimension was 260. We then produced two

copies of each image - one with a Gaussian blur of sigma 0.75 to prevent aliasing, and another without blurring. Finally, we center-cropped both the blurred and non-blurred images to the middle 256 by 256 pixels.

Train-Test Split and Class Distribution

We next performed an 80-20 train-test split on the data. We show the categories with the number of samples in our finalized data below. Our classes are not balanced and each makes up a distinctly different share, with sedans being the largest, and the others being a variety of smaller sizes.

Class	Train	Test	Total	Percent
Convertible	1,140	279	1,419	18.94%
Pickup	713	189	902	12.04%
SUV	1,616	441	2,057	27.46%
Sedan	2,534	580	3,114	41.56%
Total	6,003	1,489	7,492	

Feature Extraction

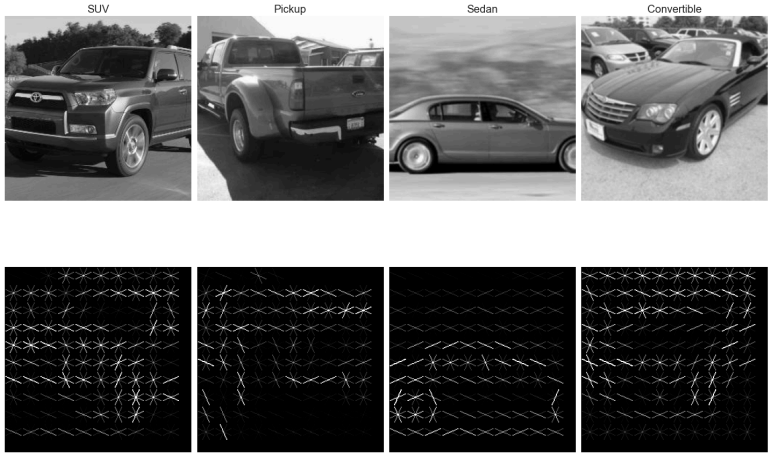
Histogram of Gradients (HOG)

In our project, capturing the structural nuances of objects was essential. To do this we applied Histogram of Gradients (HOG), a technique with the ability to extract vehicle outlines and shapes, thereby discerning between disparate body types such as sedans and SUVs. HOG achieves this by aggregating gradient orientations within discrete spatial regions—cells—rather than evaluating gradients at the pixel level. This aggregation, quantified into orientation bins and weighted by gradient magnitudes, diminishes the influence of inconsequential variances likely due to noise. Applying this method across all cells in an image provides a structured representation that encapsulates the dominant geometric features while permitting variations in shape. Thus, HOG features effectively preserve a distinct yet adaptable depiction of an object's form.

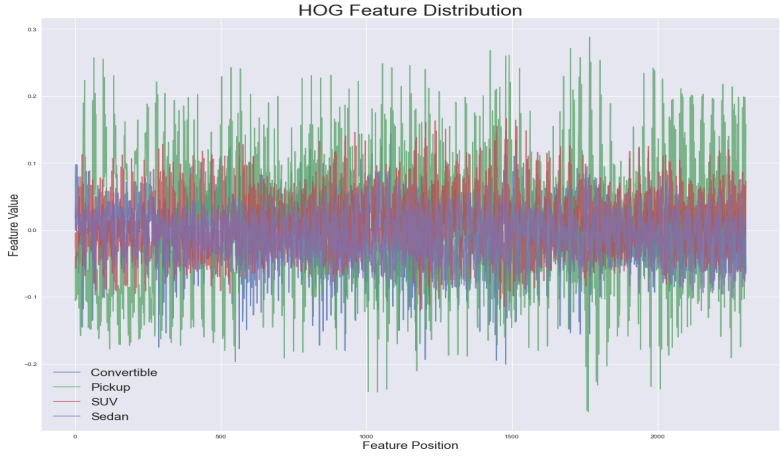
In the process of optimizing the HOG settings for our classification task, various configurations were tested to determine the balance between feature detail and computational efficiency. Our initial configuration of *16 x 16 pixels per cell*, *3 x 3 cells per block*, and *9 orientations* resulted in a finer grid of cells due to the smaller pixel dimensions per cell, allowing for a more detailed feature extraction, particularly useful for capturing subtle variations in the image such as the edges and textures of cars. These settings produced a high-dimensional feature space (15,876 features), which theoretically enhances the model's ability to learn from complex patterns in the data. However, this did not significantly enhance the model's performance in terms of accuracy or other classification metrics. The reason could be

attributed to the complexity and possible overfitting due to the exceedingly high dimensionality of the feature space. We revised our configuration to *24 x 24 pixels per cell*, *3 x 3 cells per block* with *4 orientations*, meaning our feature vectors were of dimension 2,304. This would still capture the key attributes we aimed to represent with HOG but be far simpler and more computationally tractable.

The figure below demonstrates the HOG descriptors applied to four vehicle categories: SUVs, pickups, sedans, and convertibles. The HOG visualizations reveal distinctive patterns corresponding to the structural nuances of each vehicle class. Notably, the SUV and pickup images exhibit pronounced angular contours, aligning with the typical geometry of these vehicles.



Our next figure provides a comparative analysis of the HOG feature vectors for the same set of vehicle types. The graph showcases the differential distribution of feature values across the vector space, where distinct peaks and troughs correspond to characteristic shapes and edges. The observable separation in the feature distributions underlines the discriminative capability of the HOG descriptor.



Final Project Report

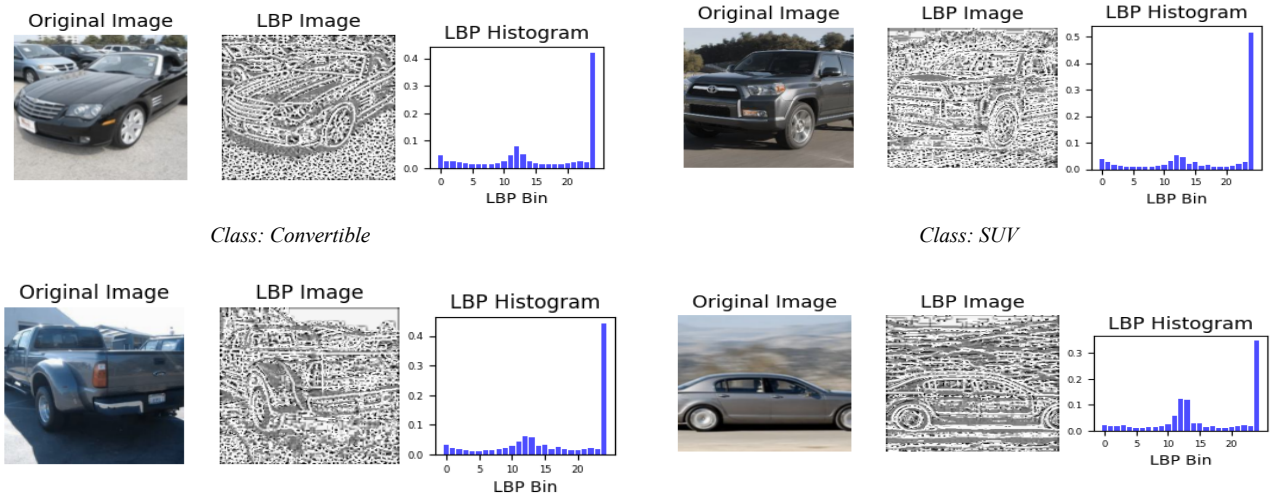
INFO 290T: Computer Vision

Local Binary Patterns (LBP)

Local Binary Patterns are a type of texture descriptor used in image processing and computer vision tasks. They describe the local patterns of pixel intensities in an image by comparing each pixel with its neighboring pixels. The patterns are then encoded into binary numbers, which are used to represent the texture features of the image.

In the context of our classification with the Stanford Cars dataset, LBP features are included to capture texture information from the images. Texture features, such as those extracted using LBP, can be valuable for identifying intricate details in the images, such as grills, bonnet design, and other car-specific characteristics. Specifically, LBP assisted in extracting car features such as the rear and headlights more effectively.

Settings for extracting LBP features typically involve parameters such as the radius of the circular neighborhood around each pixel, which we set as 3, and the number of neighboring points to consider, which we set at 24. These settings can affect the sensitivity of the LBP descriptor to different texture patterns and influence its performance in classification tasks. Our values were selected after experimentation with different combinations.



Class: Convertible *Class: SUV*

Class: Pickup *Class: Sedan*

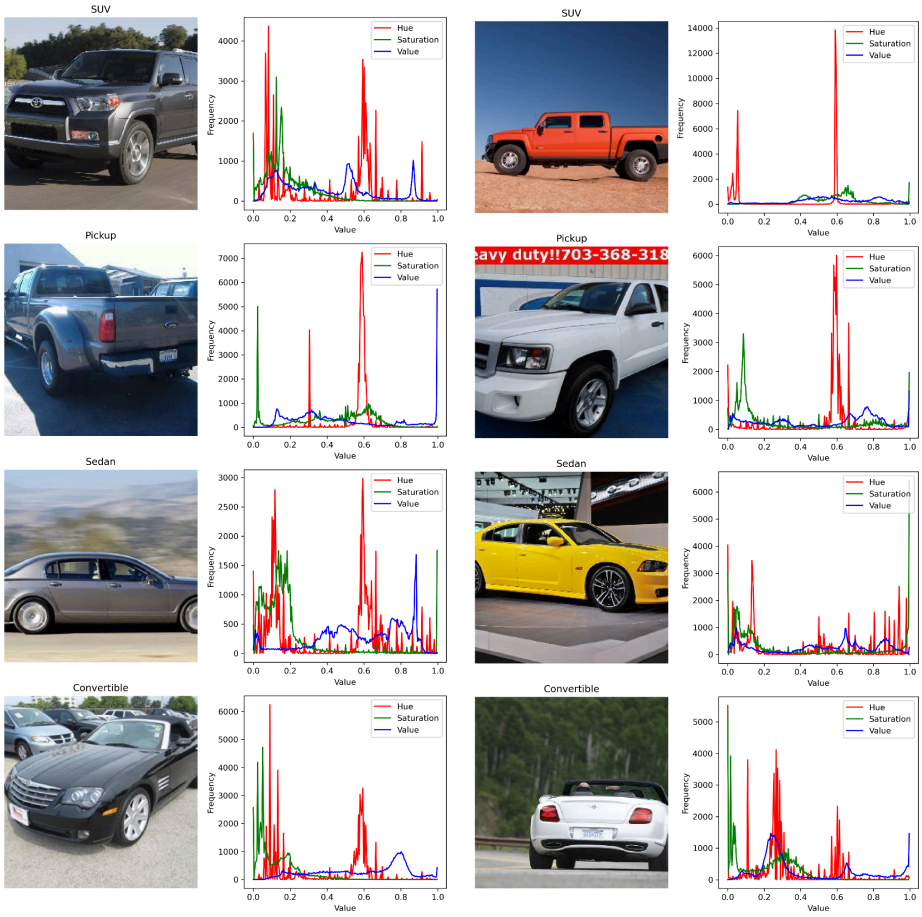
LBP Visualizations

By analyzing the LBP histogram visualizations for the different vehicle classes (SUV, Pickup, Sedan, Convertible), certain patterns can be observed:

- At each pixel location, the points usually have relatively small and consistent values. So most images tend to have a relatively homogenous texture.

- The differentiation between classes can be captured by the texture patterns and intensities mostly around the 12th bin.

Hue-Saturation-Value Color Histograms



HSV Visualization of Sample Images (256 bins)

HSV (Hue, Saturation, Value) histograms involve converting the image from the RGB color space to the HSV color space, which separates the color information (Hue) from the intensity information (Saturation and Value), increasing robustness to changes in illumination. Histograms are then computed for each of the three HSV channels, representing the distribution of pixel values across different hue, saturation, and brightness ranges. These histograms are concatenated to form a feature vector that captures the overall color distribution of the image.

Final Project Report

INFO 290T: Computer Vision

We used 256 bins (0-255) for calculating the color histograms for each HSV channel. The frequency of pixels falling into each bin was then counted.

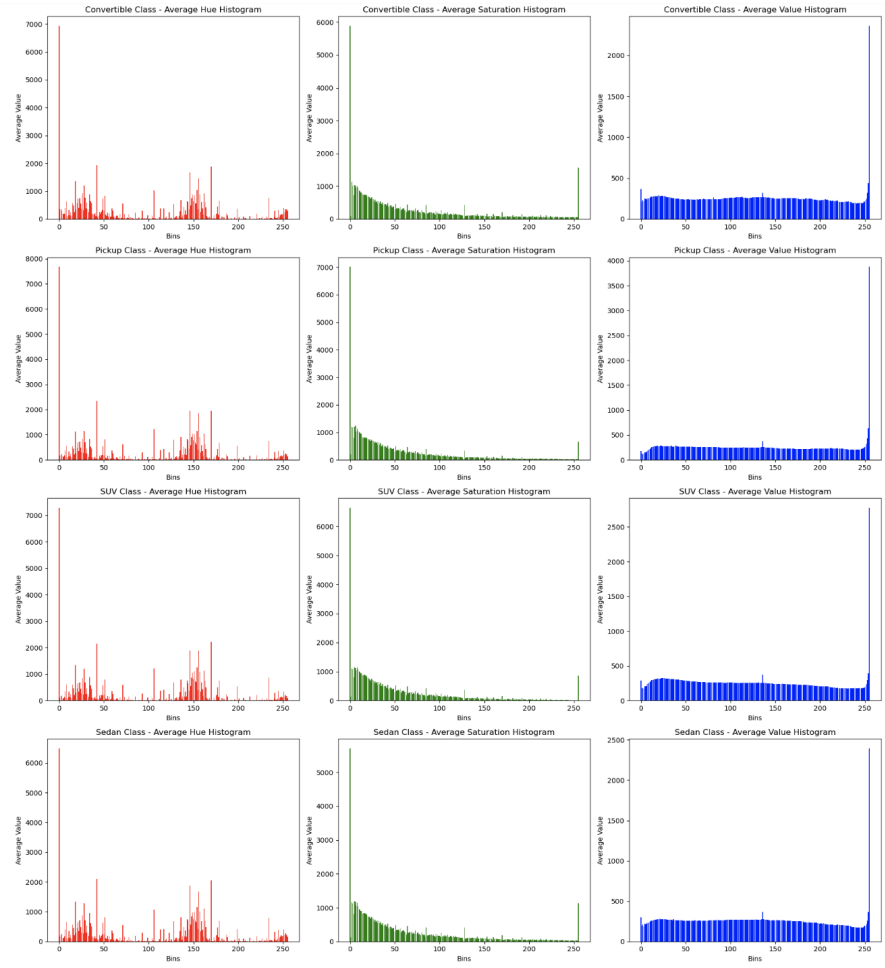
HSV color histograms were included as a feature because color distribution can be a distinguishing characteristic for certain classes of objects. For example, luxury convertible cars may tend to use more bright and exotic colors compared to other vehicle types, resulting in unique color histogram patterns.

By analyzing the HSV histogram visualizations for the different vehicle classes (SUV, Pickup, Sedan, Convertible), certain patterns can be observed:

- SUVs and Pickups tend to have higher values in the red-orange hue range (around bin 0-30), potentially indicating the prevalence of earthy tones and metallic colors in these vehicle classes.
- Convertibles often exhibit higher values in the yellow-green hue range (around bin 60-120), which could be attributed to the use of more vibrant and eye-catching colors for this vehicle type.
- Sedans seem to have a more balanced distribution across various hue ranges, suggesting a diverse color palette.
- The saturation and value channels show distinct peaks and valleys, which can further aid in differentiating between vehicle classes based on their color intensity and brightness characteristics.

Final Project Report

INFO 290T: Computer Vision



Hue-Saturation-Value Channel Average for each Class - Convertible, Pickup, SUV, Sedan

VGG

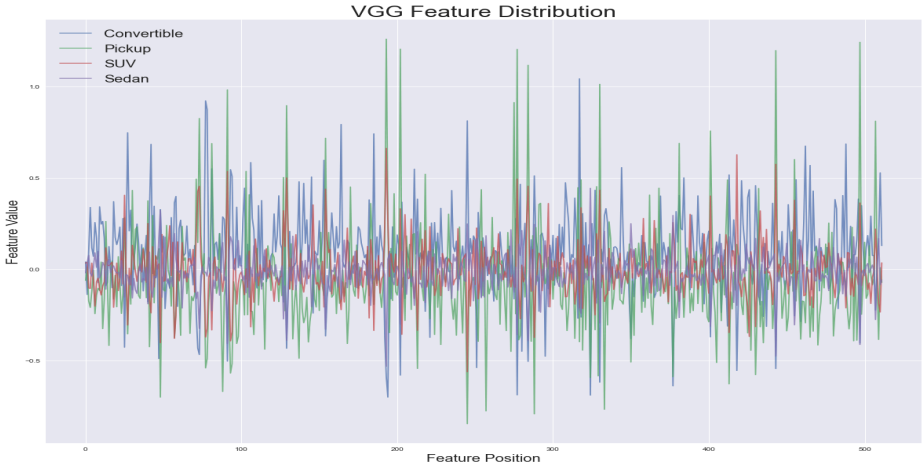
The VGG architecture, specifically the VGG16 or VGG19 models, has proven highly effective in capturing complex features from images that are useful for classification. [3] The utility of VGG arises from its deep convolutional network structure, which consists of multiple convolutional layers followed by max-pooling layers, leading to fully connected layers and a final classification layer. Its architecture allows it to capture both low-level and high-level features, making it suitable for a broad range of image recognition applications.

In our project, VGG was particularly useful due to its hierarchical feature extraction capability. The successive convolutional layers are capable of learning increasingly complex and abstract features. Early

layers may detect simple edges and textures, while deeper layers can capture high-level patterns that correspond to vehicle parts or even whole vehicles. These learned features are robust to variations in scale, orientation, and lighting, making them ideal for the nuanced task of car classification.

We included the VGG model in our pipeline to leverage its pre-trained weights, which were derived from the ImageNet dataset—a large and diverse collection of images encompassing a multitude of objects. By employing a pre-trained VGG model, we capitalized on the rich feature representations that it already possessed, a process known as transfer learning.

The figure above juxtaposes the original image of a vehicle against its corresponding feature activation map. The heatmap provides a visual guide to the regions within the image that most significantly inform the network's original classification decision. Regions with warmer colors (red-yellow) denote higher activation, suggesting these features are crucial in the identification process, while cooler colors (blue-green) represent lower activation levels.



The figure above shows the dispersion of features for different vehicle types across the model's feature space. Each line represents the feature intensity for the respective vehicle class across various positions in the feature vector. The variability within these lines reflects the uniqueness of the VGG16-extracted features pertinent to each class.

Vision Transformer

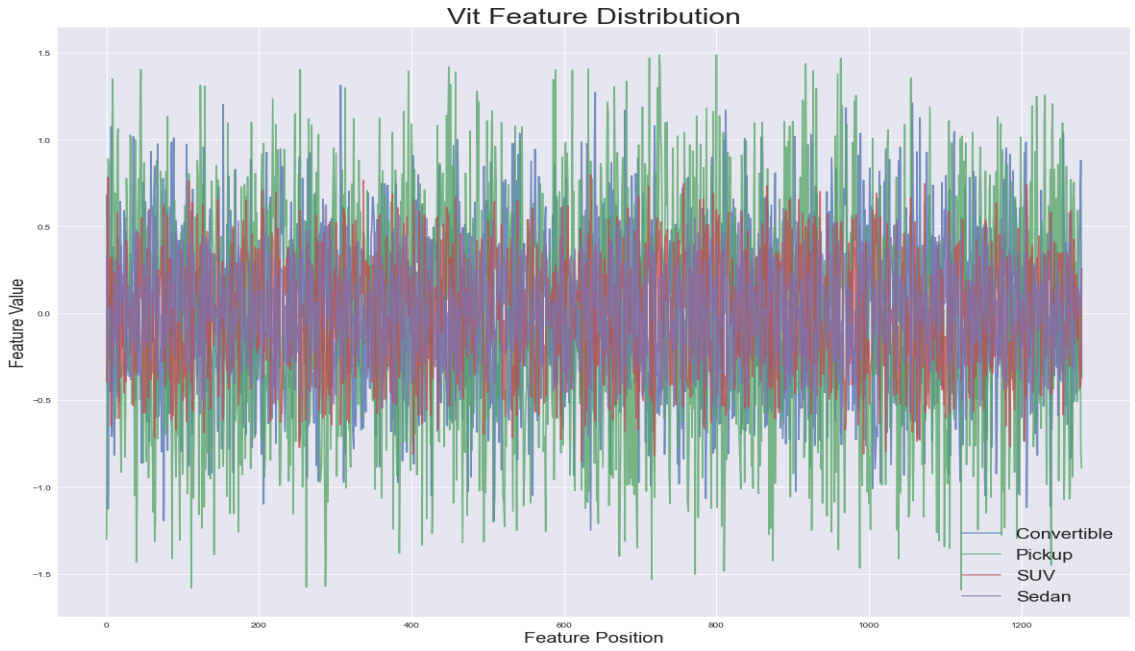
Our final feature is comprised of Vision Transformer (ViT) embeddings [4]. Vision transformers make use of the transformer architecture, now ubiquitous in large language models and natural language processing, to process images as a series of smaller patches (analagous to tokens/words in language modelling). After dividing images into patches, a linear transformation (matrix multiplication) is applied to convert the patch pixels into a new encoding vector, and another transformation is used to convert the patch position into a position encoding vector. These vectors are added together, then passed through a standard tranformer encoder, which uses attention weights indicating the strength of relationships between patches as well as fully connected multi-layer-perceptron layers to learn a useful representation.

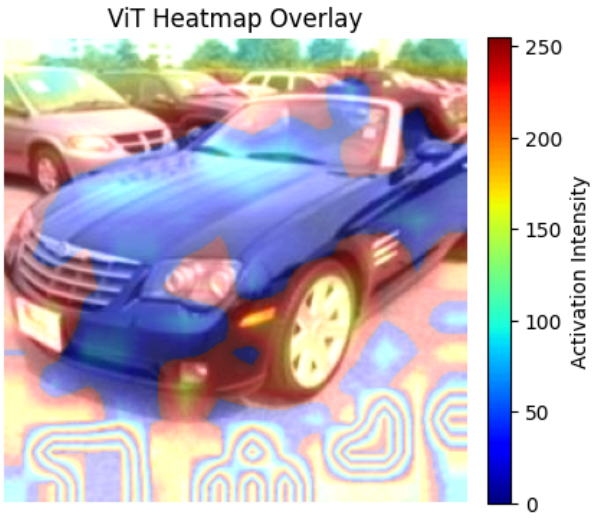
We generated embeddings from Google's *vit-huge-patch14-224-in21k* (the model in the original ViT paper), retrieved from Hugging Face. This model has 632 million parameters, makes use of 14 by 14 patches, takes 224 by 224 pixel input, and was pretrained on ImageNet 21K. We first used the provided pre-processing transformations to resize images to the appropriate resolution and normalize the RGB channels. We then passed images through the model and constructed embedding vectors as the output of the final pooling layer (a final classification layer was not included in the pretrained model). Our final embeddings contain 1,280 elements.

Including this feature brings several benefits to our project. By learning relationships between patches, vision tranformers directly focus on learning a strong global sense of ways to represent images, relative to convolutional neural networks such as VGG, which pool characteristics up from a local level in a more pre-defined fashion. Additionally, the specific model we selected was pretrained to learn representations from ImageNet 21K, which includes the classes "Pickup", "SUV", and "Convertible", providing strong off-the-shelf performance.

Final Project Report

INFO 290T: Computer Vision





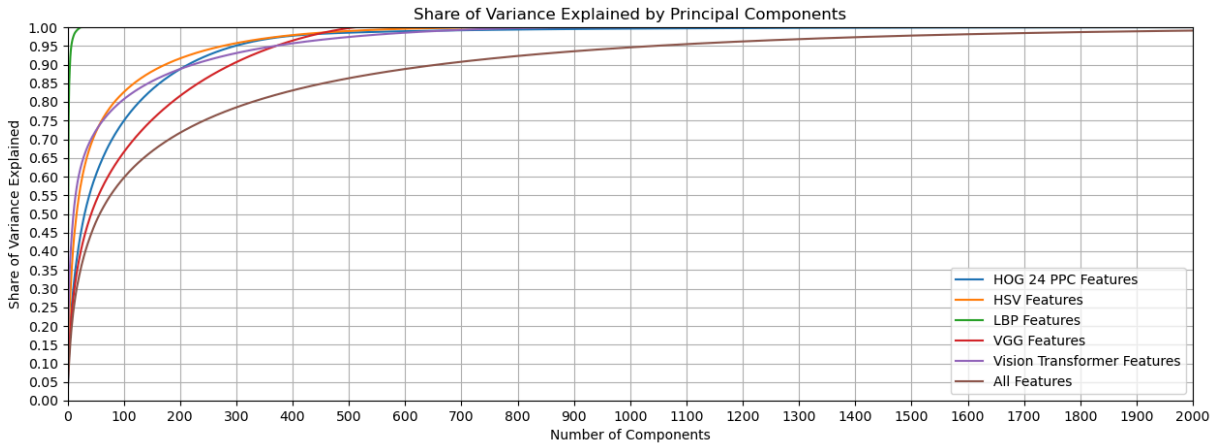
Our vision transformer feature shows a powerful amount of class variability, with distinctly different values for items in different classes. The heatmap demonstrates our model is clearly able to pick up on critical features of cars (front/grill, back/shape) in our images.

Final Project Report

INFO 290T: Computer Vision

Dimensionality Reduction

We performed principal components analysis (PCA) to linearly project our data of 4,889 columns to a lower-dimension representation that still captures maximal variance. The plot below shows the share of variance explained by various numbers of principal components on the training set. We performed this analysis for matrices comprising each of our individual features, and for a matrix combining all features.

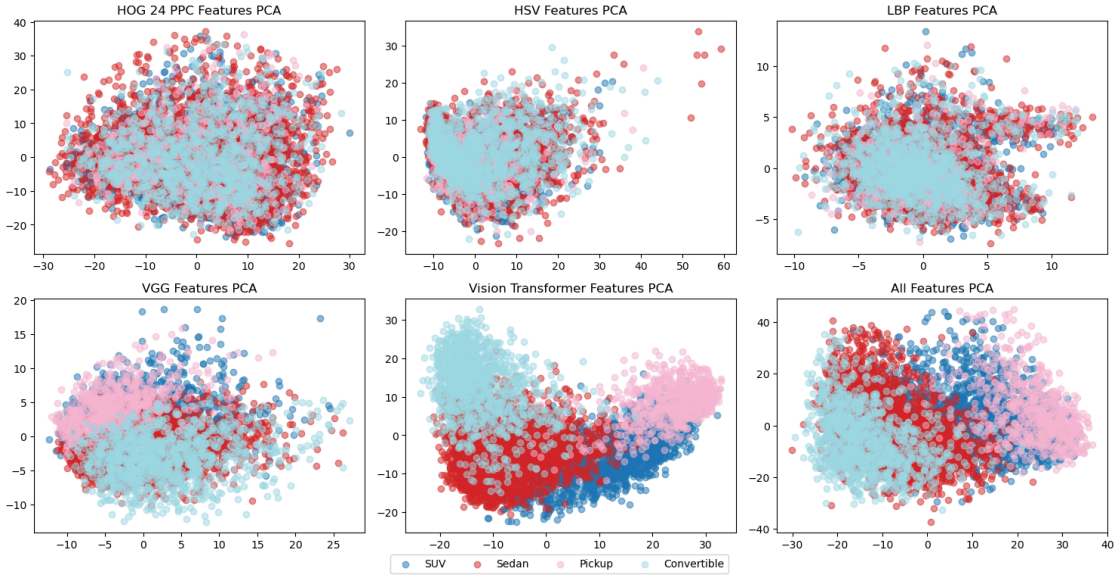


In addition to our dataset consisting of the original features, we used these train set PCA representations to create reduced dimension versions of both the training and test data. First, we created “Individual Features PCA” versions of train and test using a conservative number of components explaining at least 95% of the variance for each individual feature type: this was 10 for our LBP feature, and 400 for all others. For our HOG feature, which stood at 2,304 columns and our Vision Transformer feature, which stood at 1,280, this was a significant dimensionality reduction. Second, we performed a similar procedure to create “All Features PCA” versions of train and test using 1,250 components explaining the matrix containing all of our features.

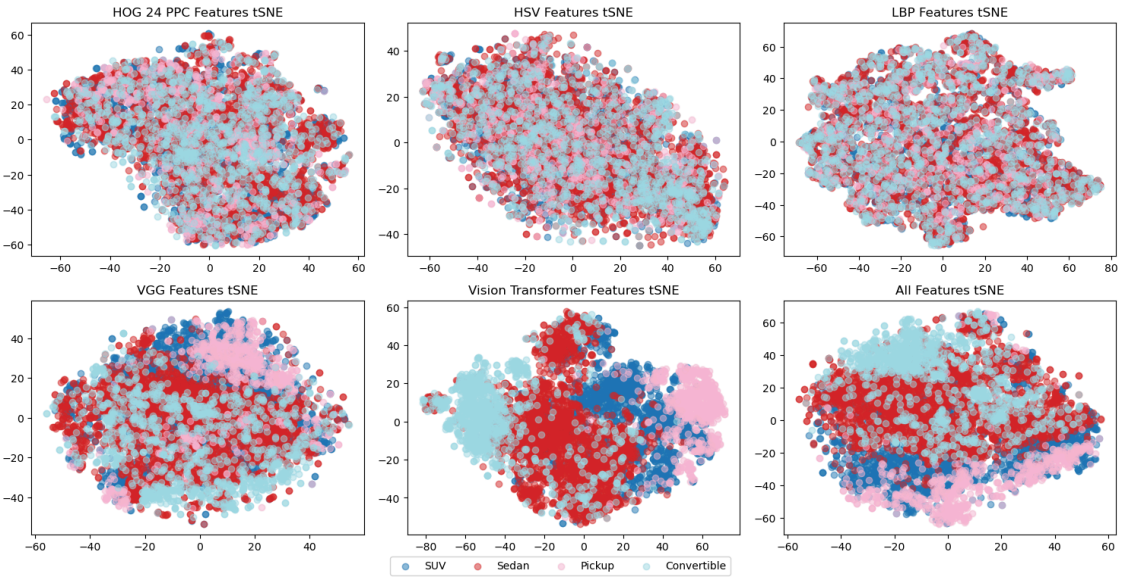
We also used PCA to transform our original dataset into two dimensions to visualize differences between classes in embedding space. Though the results are not perfect for predicting the quality of features for classification, we were able to see strong evidence that our neural features would be highly useful in separating classes even after dimensionality reduction, and that separability could be maintained when mixing the features. It was clear that even simple linear classifiers had the potential to perform well, even in this highly reduced dimension space.

Final Project Report

INFO 290T: Computer Vision



PCA provides a global linear transformation of the data, but we were also interested in other forms of dimensionality reduction. t-distributed stochastic neighbor embeddings (tSNE) provide a view of the localized differences between classes in two dimensions, and are constructed in a non-linear and iterative fashion. We were again able to see clear separability running tSNE on our vision transformer feature, while our VGG feature was somewhat less separable. Again, we were able to maintain a good degree of separability even when collapsing all the features to two dimensions.



Final Project Report

INFO 290T: Computer Vision

Classification

We implemented four different classifiers across five different versions of our feature vectors. For our classifiers, we opted to include logistic regression, a support vector machine (SVM), and tree-based classifiers with boosting and bagging - XGBoost and Random Forest. Our datasets included one with all features, versions with PCA used for dimensionality reduction as described above, a version including our "Top 3" features (LBP, HSV, VGG), and a version including VGG only. We were not able to get our SVM running on the All Features dataset due to computing resource constraints. The table below shows our model and feature combinations, the optimal hyperparameters found via grid search and five-fold cross validation, and our train, mean cross-validated, and test accuracies.

Model	Features	Tuned Hyperparameters	Train Accuracy	Mean Cross-Validated Accuracy	Test Accuracy
Logistic Regression	All Features	C: 0.1, class_weight: balanced, l1_ratio: 1.0, multi_class: ovr, penalty: elasticnet, solver: saga	0.984175	0.915708	0.910678
Logistic Regression	Individual Features PCA	C: 0.01, class_weight: balanced, l1_ratio: 0.25, multi_class: ovr, penalty: elasticnet, solver: saga	0.958021	0.915876	0.909335
Logistic Regression	All Features PCA	C: 0.1, class_weight: None, l1_ratio: 0.5, multi_class: ovr, penalty: elasticnet, solver: saga	0.992171	0.904881	0.890531
Logistic Regression	VGG	C: 0.1, class_weight: None, l1_ratio: 1.0, multi_class: multinomial, penalty: elasticnet, solver: saga	0.918041	0.863565	0.857623
Logistic Regression	Top 3 Features	C: 0.1, class_weight: None, l1_ratio: 1.0, multi_class: ovr, penalty: elasticnet, solver: saga	0.923205	0.858071	0.850235
Random Forest	All Features	bootstrap: False, max_depth: 80, max_features: sqrt, min_samples_leaf: 1, n_estimators: 1500	1.000000	0.861568	0.852921
Random Forest	Individual Features PCA	bootstrap: False, max_depth: 40, max_features: sqrt, min_samples_leaf: 1, n_estimators: 500	1.000000	0.808426	0.789792
Random Forest	VGG	bootstrap: False, max_depth: 60, max_features: sqrt, min_samples_leaf: 2, n_estimators: 1500	1.000000	0.806263	0.784419
Random Forest	Top 3 Features	bootstrap: False, max_depth: 80, max_features: sqrt, min_samples_leaf: 2, n_estimators: 500	1.000000	0.793270	0.757555
Random Forest	All Features PCA	bootstrap: False, max_depth: 60, max_features: sqrt, min_samples_leaf: 1, n_estimators: 1500	1.000000	0.698984	0.701142
SVM	Individual Features PCA	C: 10.0, class_weight: balanced, gamma: 1e-05, kernel: rbf	0.963185	0.904881	0.904634
SVM	All Features PCA	C: 10.0, class_weight: balanced, gamma: 1e-05, kernel: rbf	0.964351	0.882725	0.898590
SVM	VGG	C: 10.0, class_weight: balanced, gamma: scale, kernel: rbf	1.000000	0.869729	0.862324
SVM	Top 3 Features	C: 10.0, class_weight: balanced, gamma: 2.51e-04, kernel: rbf	0.989005	0.848908	0.838818
XGBoost	All Features	learning_rate: 0.5, max_depth: 3, min_child_weight: 1, n_estimators: 300	1.000000	0.908546	0.901276
XGBoost	Individual Features PCA	learning_rate: 0.5, max_depth: 3, min_child_weight: 3, n_estimators: 300	1.000000	0.890556	0.875756
XGBoost	All Features PCA	learning_rate: 0.5, max_depth: 3, min_child_weight: 3, n_estimators: 300	1.000000	0.860402	0.864338
XGBoost	VGG	learning_rate: 0.5, max_depth: 3, min_child_weight: 1, n_estimators: 300	1.000000	0.863733	0.854265
XGBoost	Top 3 Features	learning_rate: 0.5, max_depth: 3, min_child_weight: 1, n_estimators: 300	1.000000	0.859736	0.848892

The results for each classifier are discussed in detail in sections below. In general, the usage of PCA and lower dimension datasets tends to hurt our classifier performance. PCA applies a forced linear transformation to the data, which can obscure important information, and all of our features seem to contribute something to a properly tuned classifier. We instead found it more useful (at least in terms of accuracy) to apply dimensionality reduction via regularizing hyperparameters, optimally determined through our grid search and five-fold cross validation. Our overall hyperparameter search space is described in the table below.

Final Project Report

INFO 290T: Computer Vision

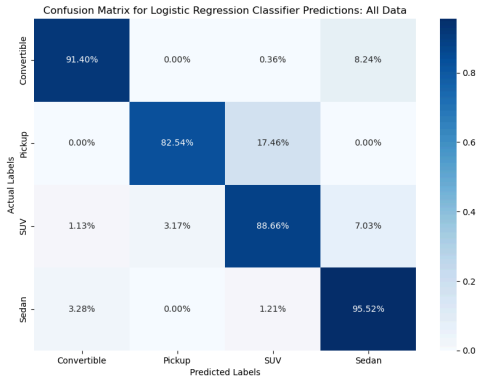
Classifier	Hyperparameter	Options
Logistic Regression	solver	saga
Logistic Regression	penalty	None, elasticnet
Logistic Regression	C	1, 0.001, 0.01, 0.1, 10, 100
Logistic Regression	class_weight	None, balanced
Logistic Regression	multi_class	ovr, multinomial
Logistic Regression	l1_ratio	0.0, 0.25, 0.5, 0.75, 1.0
Random Forest	n_estimators	500, 1000, 1500
Random Forest	max_features	sqrt
Random Forest	max_depth	20, 40, 60, 80
Random Forest	bootstrap	True, False
Random Forest	min_samples_leaf	1, 2
SVM	c_values	0.01, 0.32, 10.0
SVM	kernel_grid	rbf
SVM	gamma_grid	1e-05, 2.51e-04, 6.31e-03, 0.16, 3.98, 100.0, scale
SVM	class_weight	balanced, None
SVM	k_folds	5
XGBoost	n_estimators	100, 200, 300
XGBoost	learning_rate	0.1, 0.3, 0.5
XGBoost	max_depth	2, 3, 5
XGBoost	min_child_weight	1, 3, 5
XGBoost	k_folds	5

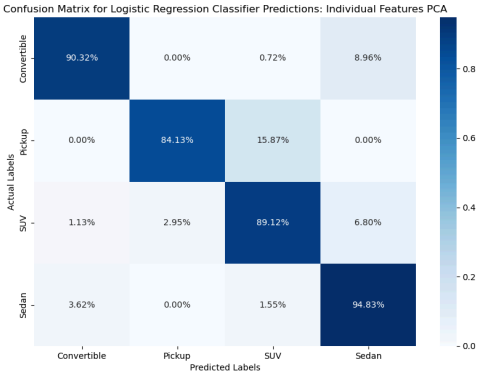
Logistic Regression

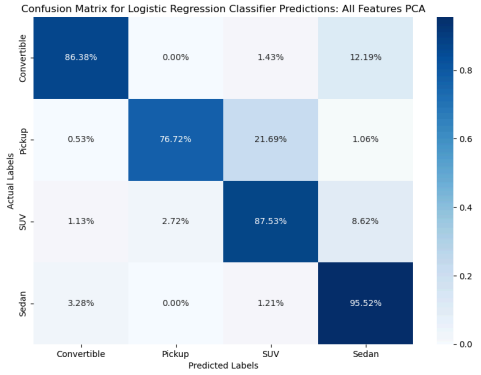
Simple logistic regression: the iterative estimation of linear weights directly applied to feature elements, then passed through a sigmoid or a softmax function, performs the best on our dataset. The model tends to perform fairly well across all classes (balanced class weighting was chosen for some models, though more could be done), and the misclassified images seem to suffer from cropping of important areas of the car (the back) and backgrounds that may mislead.

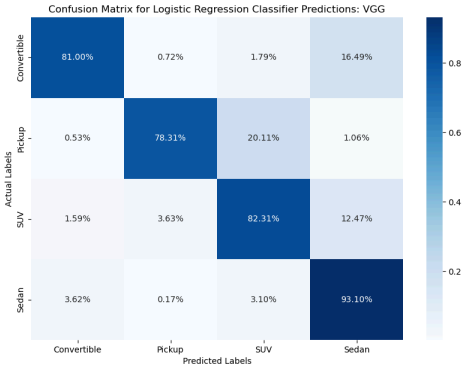
Final Project Report

INFO 290T: Computer Vision





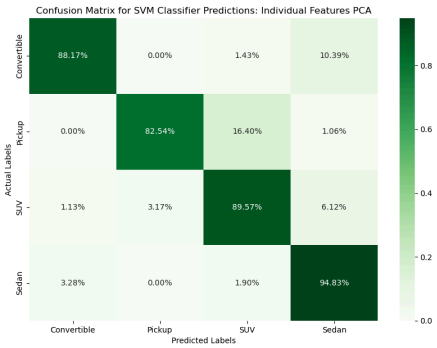


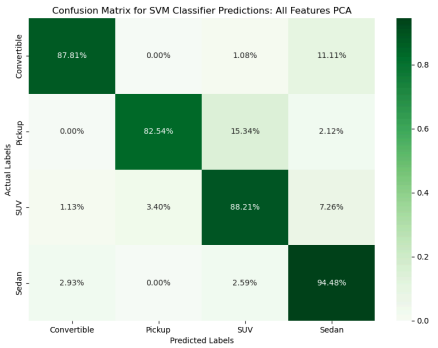


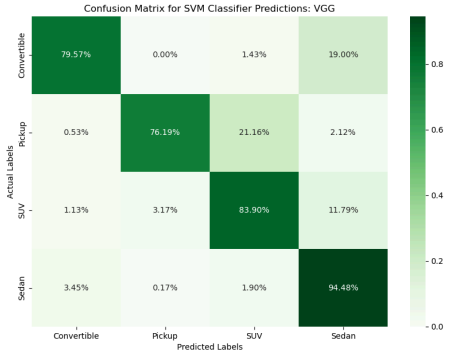
Our grid search for logistic regression parameters guided us to deploy a strong amount of L1 and L2 regularization, optimally balanced through an elastic net penalty. This effectively performed variable selection and handled our high dimensional dataset, successfully avoiding overfitting and promoting generalizability to our test set. Another potential reason for this classifier's strong performance is its

structural proximity to the linear classifier layers and softmax outputs often in the final layers of neural networks - our important neural features pair well with this setup.

Support Vector Machines







In the SVM approach, the emphasis on maximizing the margin between classes allowed for an effective delineation of feature space, especially after the application of L2 regularization. The optimal hyperparameters, identified through an exhaustive grid search, suggest an arrangement that adeptly handles the high dimensionality inherent to our vehicle dataset. The regularization parameter C is set to 10.0 owing to the complexity of the feature space, to give preference to lower bias; however, even with a smaller margin, a low Gamma value along with L2 regularization works effectively to control the variance. This balance allows the SVM to generalize well, as reflected by the comparable accuracies across train (96.32%), validation (90.49%), and test (90.46%) sets.

The confusion matrices for our SVM classifier reveal high diagonal values, indicative of a substantial true positive rate, with the model accurately predicting most of the vehicle categories. Convertibles and sedans

exhibit notably high classification rates, but in general, performance is good across all classes, possibly in part due to our balanced class weighting hyperparameter.

The principal challenge observed in misclassified cases involved partial occlusion of vehicles' defining features and complex backgrounds. Other images are truly ambiguous, even with full context. Beyond these easily explainable examples, there may be an area of feature space where differentiation between classes is not as pronounced (between frequently confused SUVs and Pickups, for example), leading to classification errors.

XGBoost

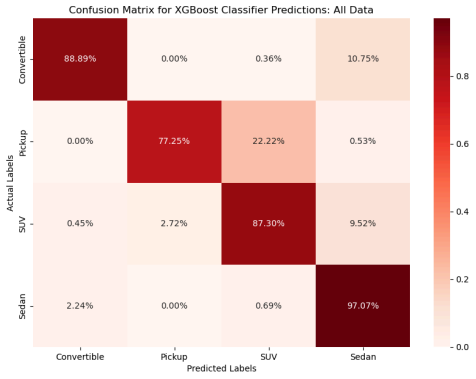
XGBoost (Extreme Gradient Boosting) is a powerful ensemble machine learning algorithm that combines multiple weak decision tree models to create a strong predictive model. It is widely used for classification and regression tasks due to its efficiency, scalability, and ability to handle various types of data, including structured and unstructured data.

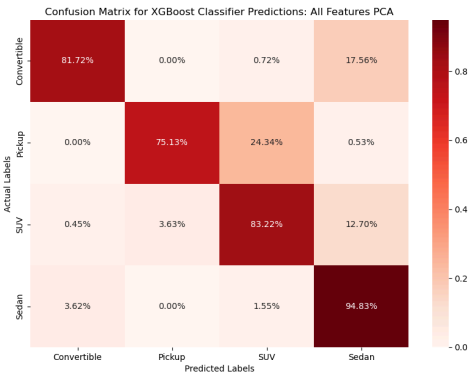
The performance of the XGBoost classifier depends on the selection of appropriate hyperparameters, which control various aspects of the model's training process. Some key hyperparameters include the maximum depth of the decision trees, the learning rate, the regularization parameters (to prevent overfitting), and the number of trees in the ensemble.

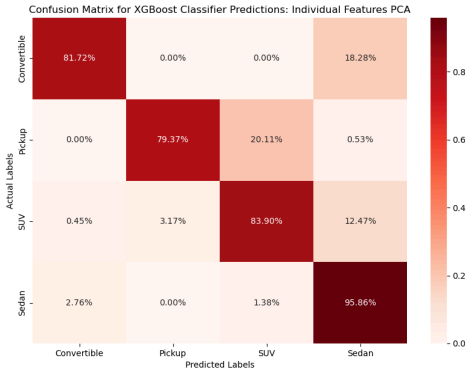
XGBoost tended to overfit on our data, exhibiting very high training accuracy and lower validation and test accuracies. This could be a result of the inherent complexity of trees. Notably, our grid search still chose maximum tree depths that were in the middle of the range (rather than lower) and minimum child weights (a regularization parameter denoting the number of samples a tree node must represent) in the lower/middle of the range (rather than higher), thus failing to prevent overfitting.

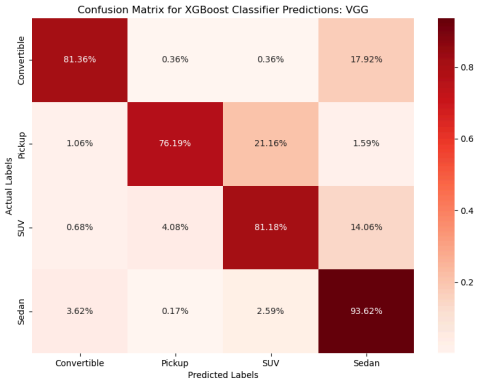
Final Project Report

INFO 290T: Computer Vision









For the All Data feature set, the classifier performs well for the Convertible and Sedan classes, with accuracies of 88.89% and 97.07%, respectively. However, it struggles with the Pickup and SUV classes, with lower accuracies of 77.25% and 87.30%, respectively. The misclassified images below show examples where the classifier confuses Pickups with SUVs, SUVs with Sedans and Convertibles, and Convertibles with Sedans.

Some of the misclassified images such as the Pickup misclassified as an SUV and the SUV misclassified as a Sedan suggest that there may be visual similarities between these vehicle types. The classifier might

be struggling to differentiate between them due to overlapping features or subtle differences that are not adequately captured by the extracted features. The misclassified images also highlight the intra-class variations within each vehicle type. For instance, the Convertible misclassified as a Sedan could be due to the classifier's difficulty in recognizing certain body styles or angles of the Convertible class. Similarly, the Pickup misclassified as an SUV might be caused by variations in the design or characteristics of certain Pickup models that resemble SUVs.

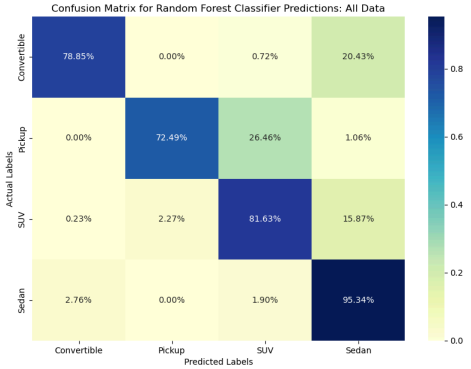
Random Forest

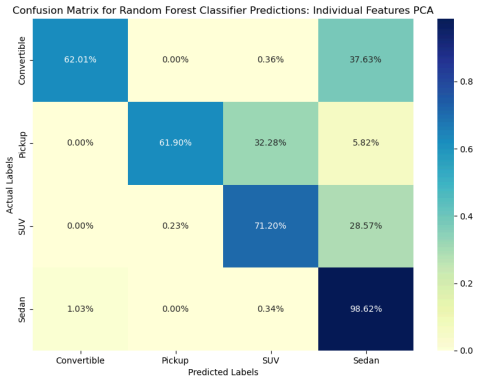
Our Random Forest classifier works by combining multiple decision trees. Each tree is built using a random subset of the dataset and features, which helps prevent overfitting by ensuring each tree is independent. This approach also helps us use the features that are most important for making predictions and identifying patterns in the data.

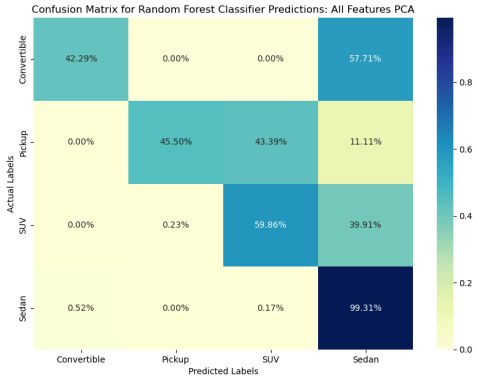
In our grid search to tune our Random Forest, we experimented with different combinations of hyperparameters, such as the number of trees and their depth. However, we were unable to prevent overfitting on the training data, meaning the model didn't generalize well. Our grid search, optimized for accuracy, often did not choose the most regularized hyperparameters most likely to prevent overfit (such as low tree depth and a high minimum samples per leaf), failing to achieve balance between model complexity and ensuring accurate predictions on unseen data.

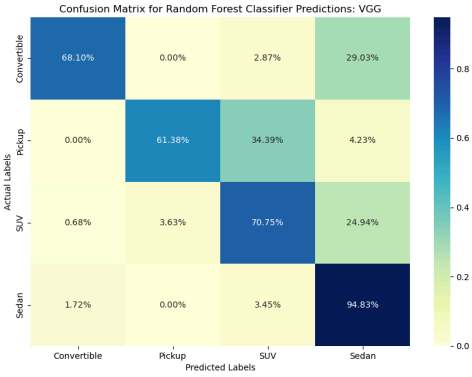
Final Project Report

INFO 290T: Computer Vision

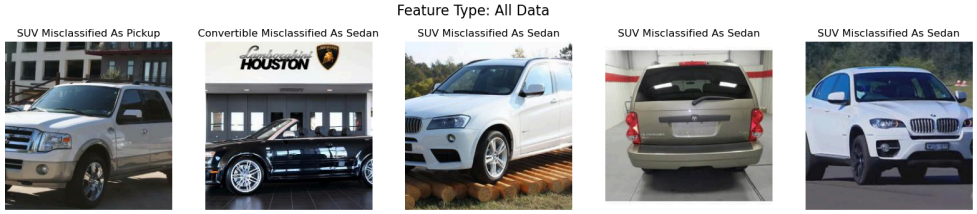








Our confusion matrices and misclassified images are shown above and below. Though we achieved a decent maximum test accuracy of 85%, items in our smaller classes (those other than Sedan) were consistently misclassified. We would have liked to investigate balanced class weighting for this model.



Final Project Report

INFO 290T: Computer Vision

Generalizability

For our logistic regression models and for most of our SVM models, we were able to attain strong results on our test dataset with relatively little degradation compared to training accuracy. This is despite the wide diversity of the images and varying angles and perspectives included. This suggests that new images could be processed through our pipeline for these models in a manner leading to fairly accurate predictions. However, as mentioned, our tree-based classifiers (XGBoost and Random Forest) appear to have overfit, achieving perfect accuracy on the training dataset and substantially lower accuracies on the validation and test datasets. This demonstrates lower generalizability that would likely be a problem with any new data, though even potentially overfit XGBoost does achieve test accuracy similar to that of our best performing logistic regression. If we were to try to fix overfitting problems, our next steps would be adjusting hyperparameter search grids to include more regularizing options, given that directly excluding features or performing PCA seemed to degrade classifier performance.

Efficiency Versus Accuracy

We performed grid search and training for our models on a variety of Intel and AMD CPUs hosted at the UC Berkeley Statistical Computing Facility (SCF). [5] Each computing job made use of 16 CPUs, but the performance of individual CPUs varies. We multiplied each CPU's Passmark score by 16 and the time taken to arrive at a measure of training workload, then normalized the results by dividing by 10,292, the Passmark of an Intel i7-1165G7 laptop CPU. [6] Hyperparameter tuning and training times for the models and features shown below are estimates for time taken on this CPU model, and all classifier inference was actually performed on this chip. Results are sorted by test accuracy.

Final Project Report

INFO 290T: Computer Vision

Model	Features	Hyperparameter Tuning + Training Time (Hours)	Inference Time (Seconds)	Test Accuracy
Logistic Regression	All Features	630.843227	0.018357	0.910678
Logistic Regression	Individual Features PCA	178.396369	0.001303	0.909335
SVM	Individual Features PCA	82.826983	14.258498	0.904634
XGBoost	All Features	174.697528	0.059770	0.901276
SVM	All Features PCA	78.335112	8.747663	0.898590
Logistic Regression	All Features PCA	141.176336	0.041061	0.890531
XGBoost	Individual Features PCA	41.979810	0.037444	0.875756
XGBoost	All Features PCA	50.466909	0.020765	0.864338
SVM	VGG	16.383645	3.604205	0.862324
Logistic Regression	VGG	84.748361	0.068139	0.857623
XGBoost	VGG	26.431934	0.011201	0.854265
Random Forest	All Features	114.034708	0.881678	0.852921
Logistic Regression	Top 3 Features	272.186602	0.062902	0.850235
XGBoost	Top 3 Features	58.524144	0.034055	0.848892
SVM	Top 3 Features	76.543556	8.010652	0.838818
Random Forest	Individual Features PCA	64.251324	0.345585	0.789792
Random Forest	VGG	48.086386	0.651461	0.784419
Random Forest	Top 3 Features	75.375151	0.213380	0.757555
Random Forest	All Features PCA	69.823949	0.894246	0.701142

Our maximum accuracy was 91.06% with logistic regression fitted on all the features with each of full dimension. Although this accuracy is over 90%, tuning and training required the equivalent of 630 hours. Performing grid search and retraining this model on new data would be cumbersome, though using it for further inference after it has already been trained is still cost-effective.

A more efficient solution is SVM fitted over VGG embeddings only, with a run time of 16 hours. A 97% percent reduction in tuning/fitting time comes at a cost of only 5% to test accuracy.

Beyond these options, the table also demonstrates some other moderately efficient solutions. One could be SVM with all features transformed via PCA, which gives 89.9% accuracy and training time of 78 hours. Although the inference time here is 8.74 secs, the huge reduction in training time is likely often worth it. Another model could be XGBoost with individual features PCA, which gives an accuracy of 87.5% with training time of just 41 hours and inference time of 0.03 secs. Considerations concerning the model's actual deployment purpose and context could help decide between the accuracy, training, and inference costs associated with these and other solutions.

Concluding Remarks

In this project, we employed a variety of image understanding techniques to create useful features to classify the type of a wide variety of car images from a large number of makes and models and a diverse

set of angles and perspectives. We were able to achieve more than 90% accuracy on our test set, demonstrating strong performance despite these challenges.

Given more time, we would have loved to improve our methods and implement more techniques. First, we would have liked to explore more hyperparameters and expand our search grids to fix overfitting and class imbalance for our tree-based models (XGBoost and Random Forest), though our accuracy for these methods often did not too seriously lag the performance of others. Second, we would have loved to implement more performant neural network models - ResNet can handle performance issues with VGG and the problem of vanishing gradients, [7] and models such as Convolutional Vision Transformers (CvT) can directly combine the strengths of convolutional networks with the attentive power of transformers. [8] We also began work on a full, end-to-end neural net classifier, and would have also liked to try our hand at implementing some sort of object detection/car size feature (possibly fitting a box or circle with expectation maximization) which might help differentiate between classes. We suspect that deployment of state-of-the-art techniques and models could lead to perfect or near-perfect accuracy on our classification problem derived from the Stanford Cars Dataset.

Final Project Report

INFO 290T: Computer Vision

Bibliography

[1] "Stanford Cars Dataset." Accessed: Apr. 17, 2024. [Online]. Available: https://www.kaggle.com/datasets/jessicali9530/stanford-cars-dataset

[2] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D Object Representations for Fine-Grained Categorization," in *2013 IEEE International Conference on Computer Vision Workshops*, Dec. 2013, pp. 554–561. doi: 10.1109/ICCVW.2013.77.

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv, Apr. 10, 2015. doi: 10.48550/arXiv.1409.1556.

[4] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv, Jun. 03, 2021. Accessed: Apr. 10, 2024. [Online]. Available: http://arxiv.org/abs/2010.11929

[5] UC Berkeley Statistical Computing Facility, "Statistical Computing Facility | Department of Statistics." Accessed: Apr. 12, 2024. [Online]. Available: https://statistics.berkeley.edu/computing/getting-started

[6] Passmark, "PassMark Software - PC Benchmark and Test Software." Accessed: Apr. 15, 2024. [Online]. Available: https://www.passmark.com

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." arXiv, Dec. 10, 2015. doi: 10.48550/arXiv.1512.03385.

[8] H. Wu *et al.*, "CvT: Introducing Convolutions to Vision Transformers." arXiv, Mar. 29, 2021. doi: 10.48550/arXiv.2103.15808.