Car Classification by Make

DATASCI 281
Abhas Wanchu, Justin Fiedler, Michael Hurth

Abstract

This project investigates vehicle make classification using classical machine learning and custom feature engineering, intentionally excluding modern end-to-end deep learning architectures. Images and labels were sourced from the Stanford Cars dataset but filtered down to 5,043 examples representing the four most common makes in the dataset: Chevrolet, Dodge, BMW, and Audi.

Feature extraction combined hand-crafted methods such as Histogram of Oriented Gradients (HOG), Canny edge detection, and Fourier transforms with deep feature embeddings extracted from a pre-trained ResNet101 model. The ResNet embeddings significantly enhanced classifier performance and demonstrated greater class separability in t-SNE and UMAP visualizations. Template matching and histogram normalization were explored but appeared unlikely to help with class discrimination. Principal Component Analysis (PCA) was applied to reduce the final high-dimensional feature set to a more manageable size for training and to mitigate the curse of dimensionality.

We evaluated two classifiers: Logistic Regression and Support Vector Machines (SVM). Hyperparameters for the models were tuned with grid search. The SVM achieved slightly better test performance (F1 score of 0.59 vs. 0.54) but exhibited substantial overfitting and high computational cost, with training time nearly 10 times that of Logistic Regression. In contrast, the Logistic model demonstrated strong efficiency with competitive performance and near-instantaneous prediction times, making it a more practical choice for scalable applications.

While both models outperformed random chance and achieved AUC scores in the “Excellent” range, overall test accuracy remained below 60%, indicating that the model is very far from commercial viability. Still, the project successfully demonstrates the value of custom feature engineering in image classification.

1. Introduction

This project aims to predict the make of a vehicle from its image. To achieve this goal, without the help of modern CNNs, we generated a series of custom features to be tested with classical machine learning classifiers. A classifier that can successfully identify vehicle makes has uses

both commercially and in the public sector. This system can be used by auto dealerships to quickly recognize inventory, or to aid in the intake of new or used vehicles into content management systems for website and sales purposes. Auto insurers can incorporate this technology into their apps to identify cars and makes for settling claims and creating quotes for prospective customers. Law enforcement can use this to help find suspect vehicles in the instances of auto theft, fugitives, and AMBER alerts.

The dataset for this car classification project is the Stanford Cars[1,2] set that consists of 16,185 images of cars and trucks compiled from various sources. The images are split out into 49 makes and 196 individual classes, comprising make, model and year. The four most common car makes, Chevrolet, Dodge, BMW, and Audi, were used as the basis for the classification task. The simplified data set contained 5,043 images to split into train, test, and validation sets at 70%, 15%, and 15% respectively. We face a slight class imbalance, with Chevrolet making up 33.5% of images in the set, with the other three makes more even across the distribution. The following table shows example images of each of the four classes and the number of images in the set.

Audi: 1,123 images	Chevy: 1,693 images	BMW: 1,024 images	Dodge: 1,203 images
			

Figure 1: Sample images

Images in the dataset are of varying resolutions and aspect ratios. The first preprocessing step was to crop all images to a square and resize the images to 224x224 pixels. An augmentation step of flipping all images was also completed to double the training set size to 7060 images. The following table shows several original examples of each class, plus the flipped version for the augmentation.

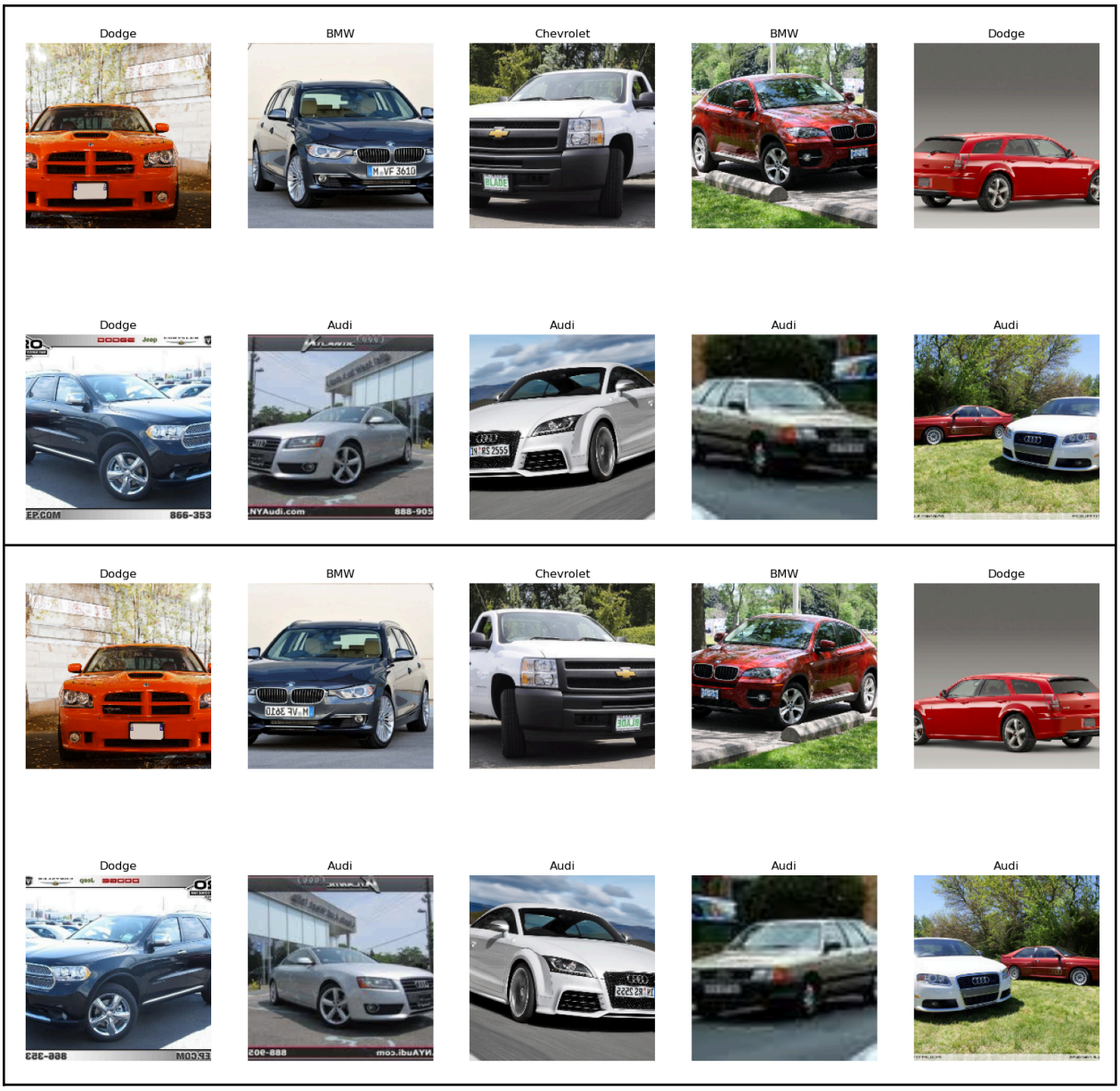

Figure 2: Example images and augmented images

Sheering, rotating, and histogram normalization were also explored in the augmentation and preprocessing phases. However, shearing was discarded because it distorted the proportions of the vehicles which are important characteristics of each vehicle class. Rotations placed the vehicles in odd orientations, which no longer resembled images the classifier would likely observe in the production environment. Shearing, as well as rotations, also introduced false edges into the image which caused artifacts in the hand generated features (HOG, Canny, and Forrier) explored below.

2. Features

Feature extraction was a critical step in developing a robust classification pipeline for distinguishing vehicle makes. Several methods were explored to identify representations that effectively captured discriminative visual patterns in the image data. Explored features include template matching, histogram of oriented gradients (HOG), canny edge detection, Fourier transforms, and pre-trained CNN embeddings.

2.1 Template Matching

One seemingly obvious distinguishing feature in the images is the brand logo. All car makes have unique emblems that get placed in various places on the car. That can include on the hood or grill, on the rear, and on wheels. It seemed a simple way to classify cars would be to just search those images for emblem matches. An early test showed promise using the *match_template* function from the sci-kit image library (figure 3).

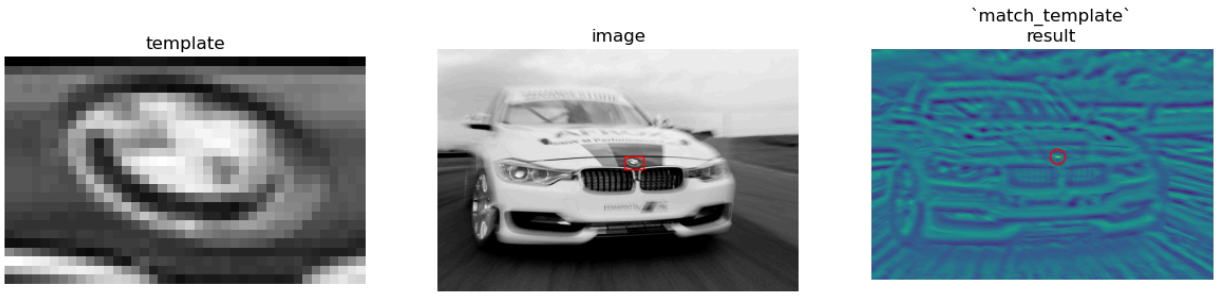

Figure 3: Template matching with template pulled from within image example

This worked well when the template was pulled from the image itself, but in practice, it was not a practical way to differentiate images in training. A standard template or set of templates for each brand would need to be utilized, and the similarity metrics used by these functions struggled to provide relevant results. In the next figure, one of these failed examples is illustrated. The *Fast Normalized Cross-Correlation* used by sci-kit image for similarity settles on a space on the windshield instead of the BMW emblem above the grill (figure 4). Experiments were also conducted using various sizes and orientations of emblems, and also with the *matchTemplate* function from the cv2 library. Because the template matching failed to correctly identify the emblems in the test images, it was not used in the final feature generation.

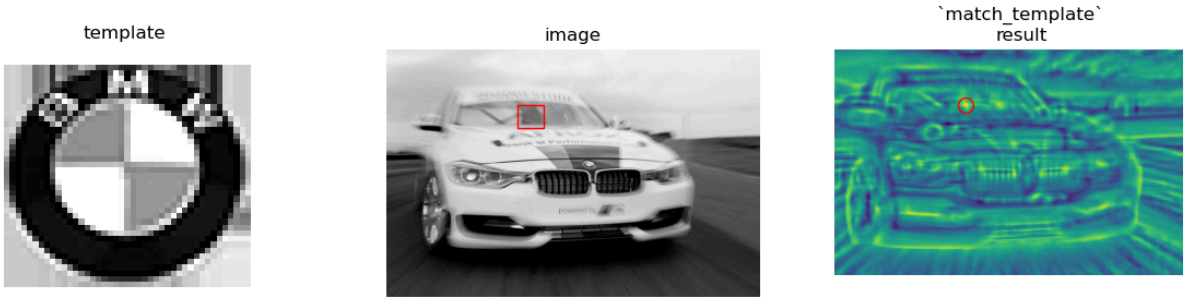

Figure 4: Template matching with standard template example

2.2 Histogram Normalization

Another image representation that was experimented with but ultimately dropped was histogram normalization, where an image's contrast is improved by altering pixel values to use the wide range of intensities. We hypothesized that normalization may emphasize the style lines of the car, the grills, and wheels. While this made some of the images in the set visually more appealing, in most cases the background noise was amplified. Extra variance appeared that was not present previously and had downstream effects on the canny edge detector. Figure 5 shows two examples. Added noise and variance would be reductive for our classification.

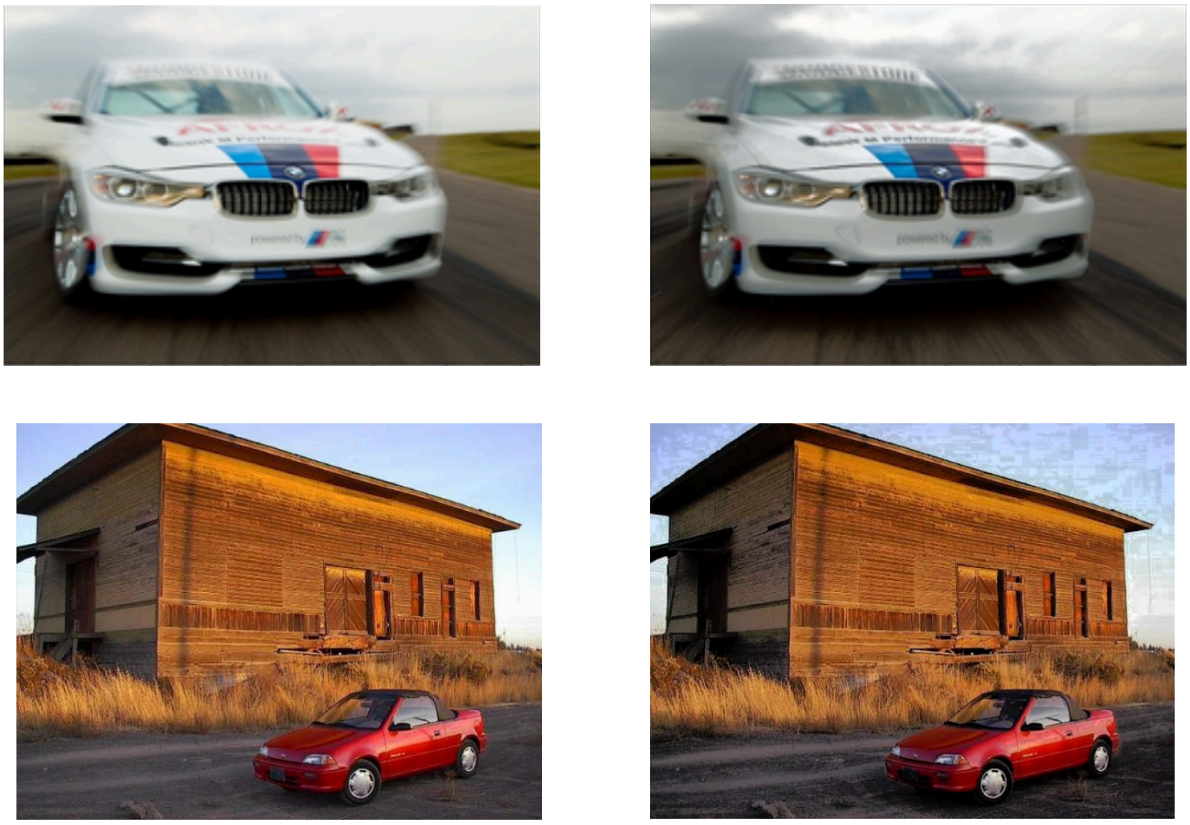

Figure 5: Example with original images (left) and histogram normalized images (right)

2.2 Canny Edge Detection

To begin building out the feature vector, one representation that was included was canny edge detection. Across vehicle brands there are many distinguishing shapes, like the shape of the car itself, style line, grills, wheels, convertible roofs, and other parts. Also, many of our images have backgrounds and other objects, and the edge detector is a way to remove some of that added information and simplify. The algorithm reduces noise, finds gradient intensity, thins down edges, and keeps or removes edge pixels by hysteresis thresholding[3]. What is left of the image is just the edges on a single color or black background. In the following set of images from our dataset, examples of canny edge detection are shown for two different classes.

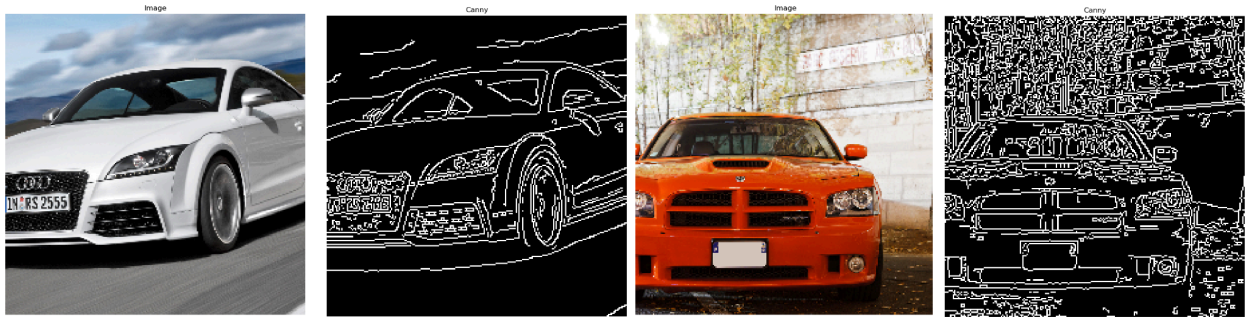

Figure 6: Canny edge detection examples

Using both a t-Distributed Stochastic Neighbor Embedding (t-SNE) plot and a Uniform Manifold Approximation and Projection (UMAP) plot, we see in the following figure that canny edge detection alone is not enough to distinguish our four classes.

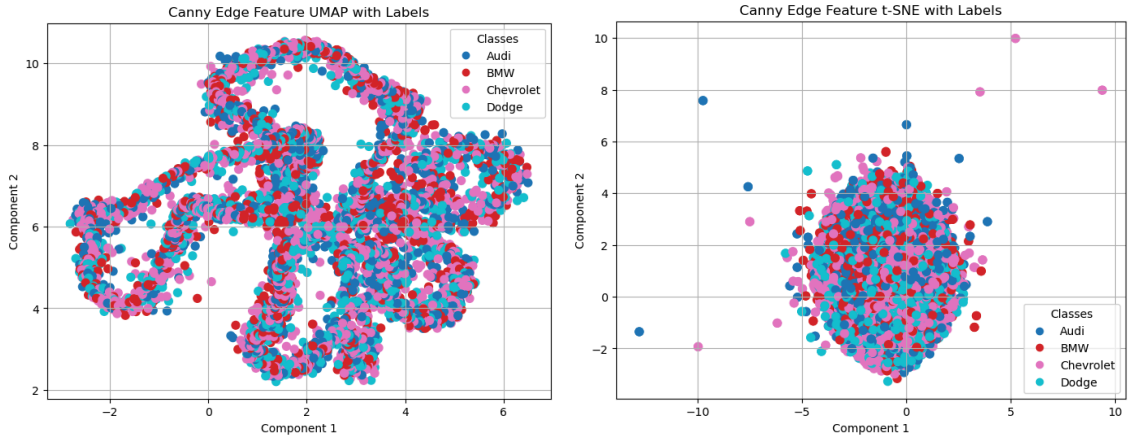

Figure 7: Canny edge t-SNE and UMAP plots showing poor class separation

2.3 Fourier Transform

With car brands sharing similar patterns (shapes, style lines, and emblems) another representation we incorporated was fourier transforms. This changes the image into its frequency domain, showing high and low frequency components. Distinct horizontal and vertical lines (like car shapes), similar structures (like vertical or horizontal grills), and repetition (like Audi's rings) can be captured in these transforms. The following images show two fourier transforms for images in the set.

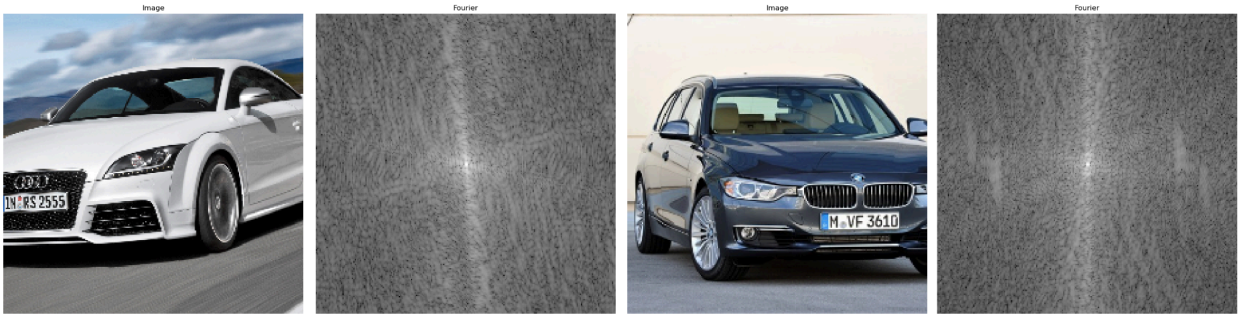

Figure 8: Fourier transforms examples

Although visually fourier transforms don't resemble the original images, it can be useful for car classification because varying types could have similar frequency signatures. Similar to canny edge detection, using t-SNE and UMAP plots, we do not see significant class separation with the fourier transforms.

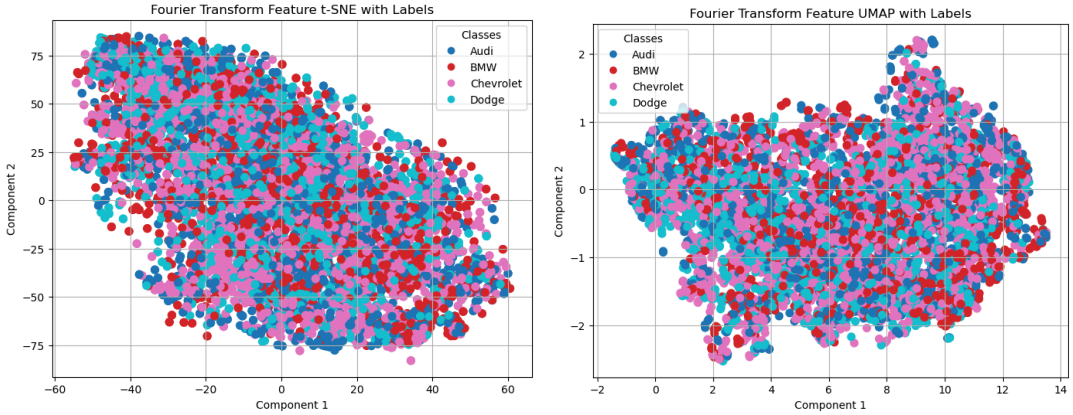

Figure 9: Fourier transform t-SNE and UMAP plots showing poor class separation

2.4 Histogram of Oriented Gradients

The final simple feature to be included in the vector for our car make classifier is Histogram of Oriented Gradients (HOG). HOG is often used for object detection, and is also a multi-stage algorithm. Images are converted to grayscale, then gradients are calculated vertically and horizontally using kernels, the image is divided into blocks, and finally a histogram is created from those gradients[4]. As can be seen in the following example images, we can see distinct features of the cars themselves, like rooflines, wheels, window shapes, and grills. These may all be distinguishing features for our classifier. In this representation, color information is discarded, which was not deemed important in determining the make of the car in each image.

Figure 10: Histogram of Oriented Gradients (HOG) examples

For this final simple feature, there is no clear separation of our four classes in the dimensionally reduced t-SNE and UMAP representations.

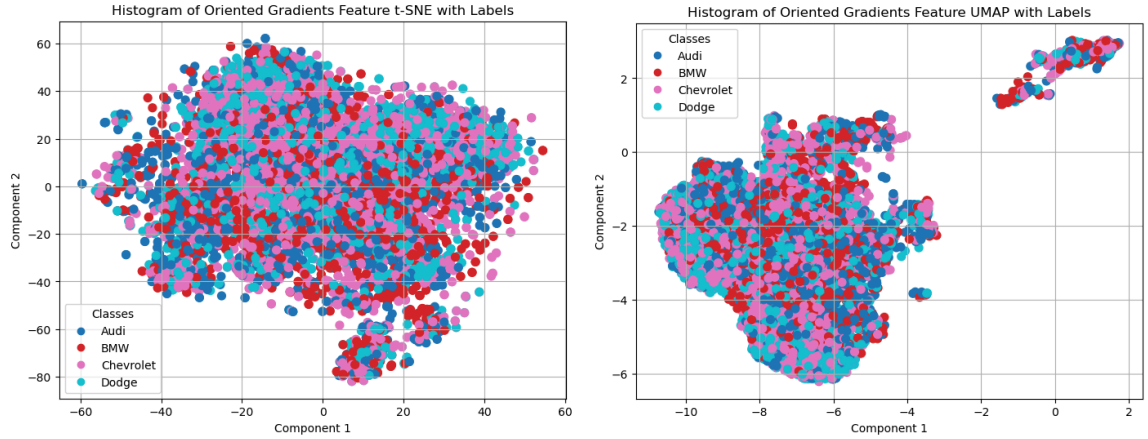

Figure 10: HOG t-SNE and UMAP plots showing poor class separation

2.5 Pre-trained Convolutional Neural Network Embeddings

The final component of our feature vector was extracted from a pre-trained CNN model. Muhib 2023[5], compared the performance of three separate pre-trained CNN architectures on the original 196 classes in the Stanford Cars dataset. Of the three models, AlexNet, VGG16, and ResNet50, he found that the Retrained ResNet50 had the best classification performance. With the findings in mind we used the ResNet model architecture, trained on the ImageNet dataset, to generate our embeddings. However, we used the deeper ResNet101 model which performs slightly better on the ImageNet classification task. Embeddings were generated by extracting the results from the penultimate layer of ResNet101 for each image. These features showed more discriminative power than the other input features as demonstrated in t-SNE plots (figure 11). In Addition, preliminary experiments with classification showed an approximately 20 point increase in weighted F1 and micro-averaged AUC scores with the addition of the ResNet101 embeddings.

Figure 11: t-SNE plot of ResNet101 embeddings showing improved class separation.

2.6 Dimensionality Reduction with Principal Component Analysis

When the aforementioned features were stacked together we had 103,968 features for each input image. With only 7060 images, the curse of dimensionality needed to be addressed before proceeding to the modeling phase. To manage high dimensionality and improve training efficiency, we applied Principal Component Analysis (PCA). However, the explained variance plot showed no clear elbow point, indicating a gradual reduction in explained variance across components (figure 12). In addition, 95% explained variance was at a feature count well above the typical heuristic used to combat the curse of dimensionality (5 times the number of samples to training features).

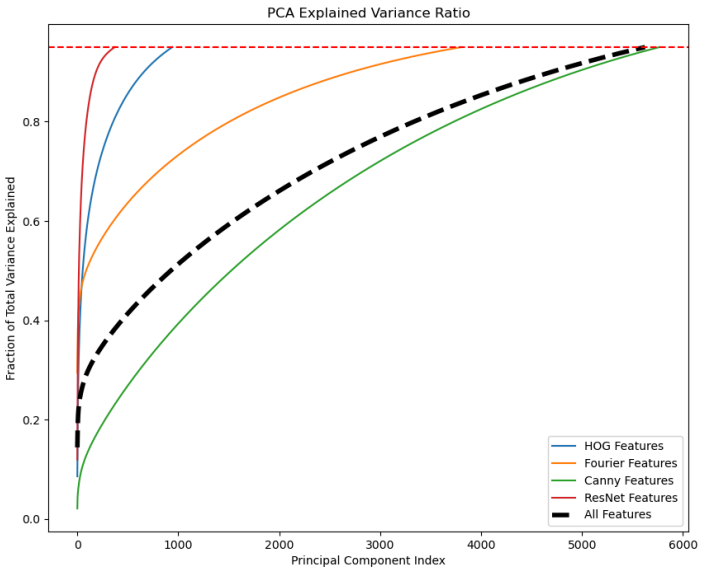

Figure 12: Feature Vector Cumulative Explained Variance Plot

To determine the optimal number of components for modeling we performed sensitivity testing with the two most promising classical machine learning classifiers from initial modeling explorations (SVM and the Logistic Classifier). Each model was trained at increasing numbers of principal component features. Weighted F1 scores and compute times were captured for each number of components for the train and validation sets. The max components in the sensitivity were limited to 3000 components given the sample size constraint. The Logistic Classifier results demonstrated that after approximately 500 components, the validation scores plateaued and the model continued to overfit (figure 13). 500 was used as the optimal component number for Logistic Regression model tuning go-forward.

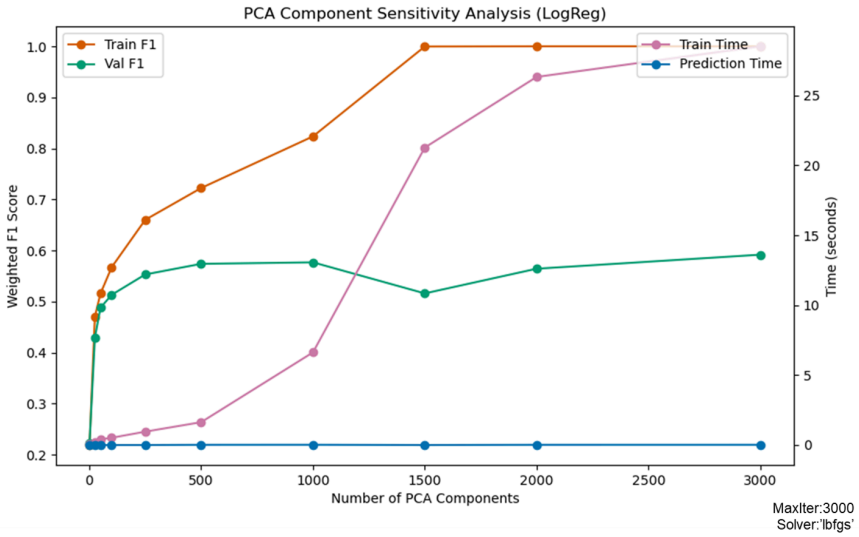

Figure 13: Principal Component Analysis vs. Logistic Classifier Performance Sensitivity

Results from the SVM sensitivity were not as actionable. While there appeared to be a distinct rollover in performance gains in the validation set, the model continued to slightly improve in performance up to 3000 components (figure 14). Since we already had a high sample-to-feature ratio and to facilitate subsequent hyperparameter search, 2000 components were used in SVM model tuning and training.

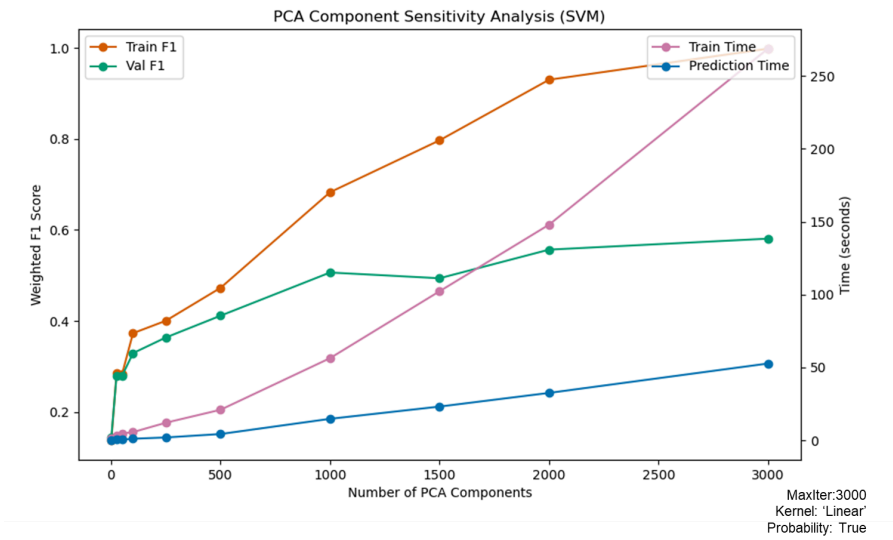

Figure 14: Principal Component Analysis vs. SVM Performance Sensitivity

3. Classification

3.1 Models

To perform our classification task, we experimented with a range of classifiers, including Logistic Regression, SVM, Decision Trees, K-Nearest Neighbors, and Random Forest. Early exploration showed that the Random Forest and KNN classifiers performed particularly poorly with a strong bias toward the majority class (Chevrolet). The SVM and Logistic Regression classifiers had the best performance on the stacked feature inputs and the PCA transformed features. Thus principal component optimization (above), hyperparameter tuning, and efficiency analysis were primarily conducted on these two classifiers.

3.2 Hyperparameter Tuning

Grid search hyperparameter tuning was performed using the training and validation sets for the SVM and Logistic Regression Classifiers. The 'Max Iter' and 'Solver' parameters were searched for the Logistic Regression. The 'Kernel' and 'C' parameters were searched for the SVM (table 1). The best parameters were determined based on the weighted F1 Score of the validation set. Optimal parameters were used to train the final SVM and Logistic Classifiers.

Model	Principal Components	Parameter	Search Values	Best Value	Best Weighted F1 Score	
					Train	Val
Logistic Regression	500	Max Iter	100, 500, 1000, 2000, 4000	1000	0.72	0.5884
		Solver	'lbfgs', 'saga'	saga'		
SVM	2000	Kernel	linear', 'poly', 'rbf'	"rbf"	1	0.59
		C	0.1, 1, 10	10		

Table 1: Grid search results for SVM and Logistic Classifiers on train and validation data

3.3 Model Results

The Logistic Regression Classifier had a weighted F1 score of 0.72 and 0.54 for the training and test sets respectively (table 2). This performance was a significant improvement over random chance (25%) or majority class prediction (33.5%). The ROC-AUC plot and confusion matrix (figure 15) show well balanced results between classes, with Audi classification performing the best and Chevrolet and Dodge the worst. AUC scores between 0.78 - 0.84 fall in the "Acceptable" to "Excellent" range.

Model	Principal Components	Params	Weighted F1 Score		
			Training	Validation	Test
Logistic Regression	500	max_iter=1000 solver='saga'	0.72	0.58	0.54
SVM	2000	kernel = 'rbf' C=10	1	0.59	0.59

Table 2: Classifier Parameters and Weighted F1 Scores

The SVM Classifier had a weighted F1 score of 1 and 0.59 for the training and test sets respectively. The SVM's test score was slightly higher than that of the Logistic Regression model, but it exhibited significantly more overfitting. Like the Logistic Classifier, the SVM ROC-AUC plot and confusion matrix (figure 16) shows well balanced results between classes. Audi has the best predictive performance, while Dodge has the worst. AUC scores between 80-85, fall in the "Excellent" range.

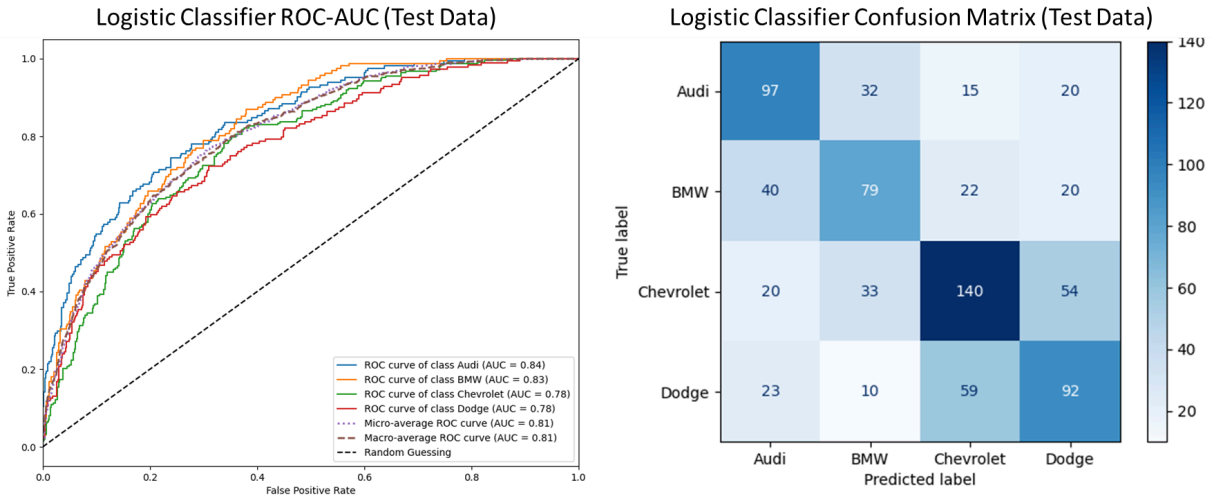

Figure 15: ROC-AUC (left) and Confusion Matrix (right) for logistic classifier on test dataset

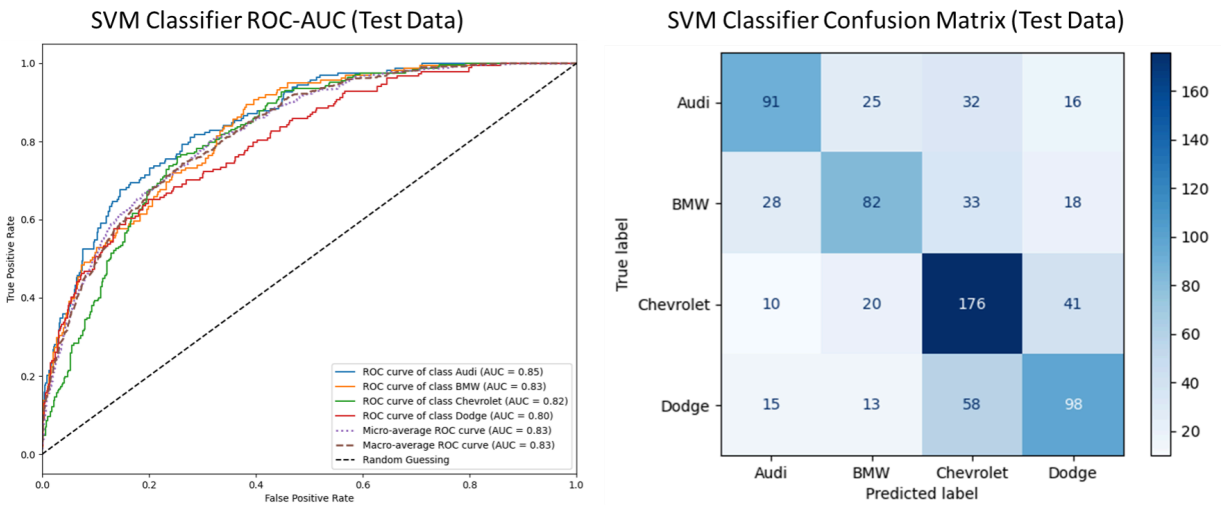

Figure 16: ROC-AUC (left) and Confusion Matrix (right) for logistic classifier on test dataset

3.4 Efficiency

Model training and prediction times were captured for the final tuned models (table 3). The SVM took 10 times as long to train as the Logistic Regression model with a training time of nearly 5 minutes. Moreover, the prediction time for the 7060 training images was nearly a minute. Conversely the logistic model prediction times were nearly instantaneous.

Model	Principal Components	Params	Model Training Time (Seconds)	Prediction Times (Seconds)		
				Train	Validation	Test
Logistic Regression	500	max_iter=1000 solver='saga'	28.25	0.0000000000	0.0000000000	0.0000000000
SVM	2000	kernel = 'rbf' C=10	288.51	53	6.36	5.47
Using 12th Gen Intel(R) Core(TM) i9-12950HX with 128GB and 24 Cores						

Table 3: Classifier training and prediction Times

4. Discussion and Conclusion

This project explored car make classification using classical machine learning methods and custom feature extraction, foregoing end-to-end deep learning. The inclusion of the hand crafted Canny edge detection, HOG, and Fourier transforms enhanced classification performance. Furthermore, the ResNet101 embeddings proved particularly effective at segregating classes, evident both in t-SNE space and in model performance. While the SVM slightly outperformed Logistic Regression (weighted F1 score of 0.59 vs. 0.54 on the test set), it came at the cost of substantial overfitting and significantly higher computational time. Although slightly less accurate, the Logistic Regression model was far more efficient. This efficiency would facilitate scaling the model to accommodate a larger number of classes and images or deploying it in a production environment, such as a mobile application.

Overall, the model performed significantly better than random guessing or majority class prediction, achieving AUC scores in the "Excellent" range. However, a test accuracy below 60% indicates it is not yet viable for commercial deployment. Key challenges for this project included class imbalance, background noise, and the high dimensionality of the feature set. Numerous avenues for future work exist, including: employing edge detection and planar homography before template matching; stratifying training images based on the market share of car makes; and investigating whether the unique paint colors employed by each manufacturer could be identified despite illumination variability in the images.

References

1. Krause, Jonathan, et al. "3D Object Representations for Fine-Grained Categorization." *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2013, pp. 554–561. https://ai.stanford.edu/~jkrause/cars/car_dataset.html.
2. "Stanford Cars." *Papers with Code*, https://paperswithcode.com/dataset/stanford-cars. Accessed 16 Apr. 2025.
3. Scikit-image canny edge detector: https://scikit-image.org/docs/stable/auto_examples/edges/plot_canny.html
4. A Gentle Introduction Into The Histogram Of Oriented Gradients: https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa
5. Uddin, Md Hasib. Evaluating the Impact of Urban Green Spaces on Mental Health Using Machine Learning Techniques. MSc thesis, University of Windsor, 2023. Scholar. UWindsor, https://scholar.uwindsor.ca/cgi/viewcontent.cgi?article=10042&context=etd.

Github: https://github.com/abhaswanchu1/mids-281-final-project-cars.git