Vehicle Detection and Classification Proposal

Jiya Varma, Madeline Liu Hou, Mengzhu Ou, Nicole Hernandez Canales, Sophia Isabel Marples Rodriguez

View on GitHub

Introduction

Vehicle detection and classification are crucial for traffic control, management, and surveillance, but manual monitoring of traffic footage is inefficient, considering the volume of vehicles in high-traffic areas. Automated vehicle classification systems can address this problem by using machine learning algorithms to detect and categorize vehicles in real-time.

Literature Review

Automated vehicle classification systems are challenging to implement, considering issues such as the computational constraints of video processing, the effect of poor weather and environmental conditions, and occlusion [1]. Researchers have focused on improving vehicle detection using machine learning classifiers and deep learning methods, such as CNN-based methods, YOLO, and Vision Transformer (ViT) techniques [1], [2], [3].

Dataset Description

We used the Stanford Cars dataset for training our models. The dataset contains 16,185 images across 196 different classes of cars, with with annotations describing the make, model, and year of the car.

We reduced the number of classes from 196 to 48 classes, which involved condensing the dataset by grouping individual car models into broader brand-based categories. Instead of treating each unique model, such as "BMW X3," "BMW X4," or "BMW E30," as separate classes, we grouped all these models under a single class, "BMW." This same approach was applied across all other car brands in the dataset.

- Dataset Link

Problem Definition

In our project, we seek to perform multi-class car classification on car images from various brands such as BMW, Acura, Honda, etc. As previously covered, the use cases of car

classification ranges from law enforcement to traffic control. Our project seeks to improve upon existing methods.

Gantt Chart

Gantt Chart Link

Implemented Methods

This section describes the implemented methods for analyzing and classifying the condensed Stanford Cars dataset. Three approaches were explored: a convolutional neural network (CNN) based on a Residual Neural Network architecture, Visual Transformer (ViT) model, and various clustering methods implemented using Gaussian Mixture Models (GMM) and other algorithms.

We will resize all images to ensure consistent input dimensions, as neural networks require uniform image sizes for layers like convolution. Data augmentation (e.g., rotation, flipping, noise injection and reduction, cropping) will be applied to generate additional images, increasing the dataset size and preventing overfitting [4]. Pixel normalization is used to standardize lighting conditions and ensure consistent image scales.

The Vision Transformer (ViT) requires “patches” from the input image, and thus a patch-creation pre-processing step will be required [5]. We plan to train 2 supervised models, Convolution neural networks (CNN) and Vision transformer (ViT), and one unsupervised model, Gaussian Mixture model (GMM), for car classification.

- CNNs are effective in image classification due to hierarchical feature detection (e.g., edges, wheels, chassis detection) and translation invariance (e.g., different angles/location of car in images) when trained on a strong dataset [7].
- ViT, which sequentially processes image patches using self-attention, has shown comparable performance to CNNs when trained on large datasets [6].
- GMM will reduce our dataset's class size of car makes/models to fewer clusters of chassis shapes (e.g., sedan, coupe, SUV), creating a more manageable dataset for training.

CNN - Residual Neural Network

For our first model implementation, we decided to create a Residual Network Model, or ResNet, based on Microsoft Research's 2015 publication. The core concept of the ResNet model is the residual block, which creates a direct connection from some input to the output of a deeper convolution layer, essentially "skipping" computations in between. This is useful in preventing vanishing or exploding gradients that often occur in deeper architectures due to high frequency of multiplications. The skip connections ensure that the gradients are kept secure from becoming too small or large, and allow for an easier flow path during backpropagation so that the network can more effectively adjust the weights. This is effective for producing a generalizable model as it can learn patterns across deeper layers without getting lost [8].

Our residual block contains 2x convolution (with rectified linear unit, or ReLU activation) + batch norm blocks such that the input tensor is added to the result of the second convolution output, and then finally passes through a ReLU activation before being returned.

The current architecture of our ResNet Model is as follows: [input tensor -> data augmentation layers -> initial convolution -> batch norm -> activation -> residual block 1 -> residual block 2 -> residual block 3 -> global average pooling -> flatten -> dense layer (with softmax)] [10],[11], [12]

We increase the filter size between each residual block to capture more complex features of our car image dataset, (theoretically) such as chassis shape or car brand logos [9]. To offset the computational cost of increased number of feature maps and maintain uniform dimensions, we employ a projection or bottleneck convolution layer with a 1x1 kernel size that ensures that the number of channels/depth of the input matches the # of channels of the residual block's second convolution's output [15].

We also employ Sparse Categorical cross entropy as our loss function as it allows direct indexing of class labels rather than requiring one-hot encoding, which is helpful for our pipeline as we have a large class size of 48 [14]. Additionally, we opted to use 'adam' (adaptive moment estimator) as our optimizer as it allows more flexibility in adjusting learning rates based on how large/small the gradients are. We have found that 'adam' is a good first choice due to its faster convergence speed and applicability to several problems without extensive need for hyperparameter tuning [13]. However, we plan to run several hyperparameter tuning experiments with various optimizers, learning rates, momentum decay, etc in order to improve our model performance.

GMM - Gaussian Mixture Model

This section explores four distinct approaches undertaken to classify and understand the Stanford Cars dataset using a combination of supervised and unsupervised machine learning methods. The first approach compares the clustering capabilities of Gaussian Mixture Models (GMM) and K-Means, utilizing silhouette scores and other metrics for evaluation. The second approach extends the unsupervised pipeline by introducing DBSCAN and Spectral Clustering to address non-spherical and arbitrarily shaped clusters. The third approach integrates a hybrid K-Means + GMM pipeline to refine clustering by leveraging the strengths of both methods. Finally, the fourth approach combines supervised models—Random Forest and Support Vector Machines (SVM)—with the unsupervised methods, enabling a direct comparison of supervised and unsupervised learning paradigms. Each approach is presented with details on pipeline structure, design choices, and the reasoning behind key decisions.

Approach 1: GMM vs K-Means - model_997376_100-iters_150-pca-components
This approach focuses on implementing Gaussian Mixture Models (GMM) and K-Means to cluster the dataset into groups that share common features, such as chassis shape or overall dimensions. The goal was to explore the efficacy of GMM in providing cluster structures while leveraging silhouette scores and other metrics for evaluation. GMM was chosen for its ability to model data with mixed Gaussian distributions, a property suitable for complex datasets like cars [16].

Pipeline Structure and Flow
Each pipeline consists of four main components: Data Loading, Clustering, Metrics Evaluation, and Visualization. The *data_loader.py* preprocessed the dataset by normalizing image data and applying PCA with 150 components, reducing computational complexity and noise. The *gmm_pipeline.py* implemented clustering using Gaussian Mixture Models (GMM) and K-Means, iterating over different cluster sizes (*k*) and covariance types. Metrics evaluation included accuracy, precision, recall, F1 score, Fowlkes-Mallows score, and silhouette score. The results and visualizations (e.g., t-SNE and confusion matrices) were stored for qualitative and quantitative analysis.

Key Design Choices and Reasoning

Key decisions included using PCA for dimensionality reduction, which balanced retaining dataset variance while reducing complexity, and employing multiple covariance types in GMM (*full*, *diag*), allowing flexibility in cluster shapes. The Hungarian Algorithm was used to map predicted clusters to ground truth labels for meaningful metric computation [17]. The inclusion of silhouette scores provided insights into cluster cohesion and separation.

Approach 2: DBSCAN and Spectral Clustering - model_998281_100-iters_50-pca-components

In this approach, additional clustering algorithms were introduced, including DBSCAN and Spectral Clustering, alongside GMM and K-Means. DBSCAN was selected for its ability to handle arbitrary cluster shapes and detect outliers, while Spectral Clustering was included for its ability to cluster non-linearly separable data. This approach further sought to refine the dimensionality reduction step with PCA, reducing the components to 50 for improved efficiency [18].

Pipeline Structure and Flow

This iteration of the pipeline extended clustering capabilities by integrating DBSCAN and Spectral Clustering in addition to GMM and K-Means. The *data_loader.py* script maintained the preprocessing steps but reduced PCA components to 50 to improve computational efficiency. The *gmm_pipeline.py* explored various clustering parameters, such as DBSCAN's *eps* and *min_samples*, and GMM's covariance types (*full*, *tied*, *diag*, and *spherical*). Metrics evaluation and visualization components remained consistent, allowing direct comparisons between clustering methods.

Key Design Choices and Reasoning

The inclusion of multiple clustering algorithms aimed to address the limitations of GMM and K-Means in handling overlapping or non-spherical clusters. DBSCAN was particularly chosen for its ability to detect arbitrary cluster shapes and identify outliers, with parameters *eps* and *min_samples* dynamically adjusted for experimentation. PCA was reduced to 50 components to further streamline computations, balancing variance retention and dimensionality reduction. Visualizations were leveraged to qualitatively evaluate the separability of clusters across algorithms.

Approach 3: Hybrid K-Means + GMM - model_998850_100-iters_50-pca-components

This approach combines the strengths of K-Means and Gaussian Mixture Models (GMM) into a hybrid pipeline. K-Means is employed for pre-clustering, initializing centroids that are subsequently refined by GMM. This two-stage approach is designed to mitigate initialization challenges in GMM and improve overall clustering performance. The pipeline leverages PCA for dimensionality reduction, allowing efficient computation while maintaining key data features. This hybrid method addresses the limitations of standalone clustering algorithms, particularly in datasets with overlapping clusters or non-spherical distributions [19].

Pipeline Structure and Flow

The pipeline begins with the preprocessing step, where PCA reduces the dataset to 50 principal components. This dimensionality reduction preserves the dataset's variance while reducing computational complexity. Following PCA, K-Means is utilized as a pre-clustering method to identify initial centroids. This step ensures a stable starting point for the subsequent GMM refinement by addressing the initialization sensitivity found in GMM. The centroids derived from K-Means are used as the initialization points (`means_init`) for GMM, allowing the algorithm to converge faster and model more complex cluster shapes. The entire pipeline is automated to store results and visualizations in structured directories, facilitating an efficient comparison across different configurations.

Key Design Choices and Reasoning

K-Means is computationally efficient and provides a deterministic clustering result, which is really good as an initialization method for GMM. GMM, in turn, refines these clusters by using its probabilistic framework. Dimensionality reduction through PCA remained at 50 components to ensure that the computational load remained manageable while retaining significant variance in the dataset. Multiple covariance types—*full*, *diag*, *tied*, and *spherical*—were explored to allow the model to adapt to various cluster shapes and data distributions [20]. The iterative refinement enabled by this hybrid approach not only improves the quality of the clusters but also ensures that the initialization step from K-Means mitigates GMM's sensitivity to random starts. Metrics and visualization components were integral to validating the pipeline's effectiveness, ensuring that results were both quantitatively robust and qualitatively interpretable [21].

Approach 4: Random Forest & SVM - model_999191_100-iters_100-pca-components
This approach incorporates supervised learning methods (Random Forest and Support Vector Machines) alongside unsupervised clustering techniques (K-Means and GMM). By evaluating both supervised and unsupervised approaches on the same dataset, the approach explores their comparative strengths and weaknesses. The supervised models aim to maximize predictive accuracy using labeled data, while the unsupervised methods explore inherent cluster structures within the dataset. The pipeline emphasizes a comprehensive evaluation of these models under a consistent preprocessing framework.

Pipeline Structure and Flow
The pipeline begins with data preprocessing using PCA to reduce dimensionality to 100 components, which balances computational efficiency with data variance preservation. For supervised learning, the preprocessed dataset is split into training and testing subsets. Random Forest and SVM models are then trained on the labeled training data and evaluated using the testing set. In parallel, the unsupervised methods—K-Means and GMM—are applied to the preprocessed data, exploring various configurations of cluster sizes (k) and covariance types for GMM. The clustering results are evaluated using metrics such as silhouette scores, Fowlkes-Mallows scores, and visualizations like t-SNE plots and confusion matrices.

Key Design Choices and Reasoning
Random Forest was chosen for its robustness to overfitting and ability to handle large datasets with minimal parameter tuning, while SVM provided a linear baseline to compare against the more complex clustering results. PCA was extended to 100 components in this approach to ensure that sufficient data variance was preserved for the supervised models, which typically rely on higher-dimensional representations. We maintained the integration of hybrid K-Means + GMM methods to combine the computational simplicity of K-Means with the flexibility of GMM to model overlapping and non-spherical clusters.

ViT - Vision Transformer

Architecture Description

The Vision Transformer (ViT) was introduced by Google Research in 2021 [22] and is an application of the transformer architecture (typically used for NLP tasks) for computer vision use cases, such as image classification. The ViT architecture requires a few data preprocessing steps such as splitting images into fixed-sized chunks, flattening and applying a linear projection to the patches to create patch embeddings, and finally adding 1-d learnable positional embeddings to the patch embeddings. This embedding vector is given to the transformer's encoder as input. The encoder contains alternating multihead self-attention blocks (which captures relationships between image patches) and multi-layer perceptrons (which introduces non-linearity to extract complex information within each patch independently). Specifically, the MLPs use Gaussian Error Linear Unit (GELU) as activation between two layers. Layer normalization is applied before each block and is used to normalize

each sample independently across its features. Skip connections, as in the ResNet model, are included after each block. For our project, we have modified an open source ViT implementation [23] to suit our car classification use case. This architecture follows the aforementioned description followed by a 1-d global average pooling layer and an output MLP classification head.

Experiments and Results

We experimented with patch sizes of 16 x 16 and 8 x 8, expecting the smaller patch size to result in higher accuracy due to containing higher resolution and details of the image. However, we found that the opposite was true.

Overall, the test accuracy from the ViT was significantly lower than the ResNet, despite adding additional data augmentation layers prior to image embedding step. This may be due to the fact that several of our images have large white borders, possibly resulting in patches of purely white pixels that can introduce disruptive noise and positional bias to the encoder. These pixels could also explain the seemingly contradictory increase in accuracy from 8 x 8 to 16 x 16 patches, since by covering more of the image, 16 x 16 are less likely to be dominated by noise.

In an attempt to address this issue, we found the median height and width of all images, then resized our images, introducing padding as needed to preserve aspect ratios. This significantly reduced the amount of padding and overall noise in each image. Images were resized to 640 x 424, then halved to 320 x 212 to reduce computational cost. Additionally, we applied a slight compression to 320 x 208 to accommodate the use of 16 x 16 patches.

It should be noted that images for ResNet were resized prior to randomly splitting the dataset into training, testing, and validation sets. As a result, resizing images for ViT and randomly sorting images into the 80-10-10 split led to differences in the distribution of images in each subset. However, these differences are unlikely to have a significant impact on accuracy since the random assignment should ensure that the splits are still representative of the overall dataset. For consistency in future comparisons, resizing the images after performing the 80-10-10 split would avoid this issue entirely.

With the updated dataset, we observed a slight improvement in accuracy. However, regardless which dataset it was trained on, the model continued to exhibit the same behavior of predicting the same class for all images. This class would also change on each run. This behavior could be attributed to unbalanced classes in our dataset, with certain classes being overrepresented, leading to potential bias towards overrepresented classes [26]. In order to account for overrepresented classes, we added class weights to our model [22]. Despite this addition, our model continued to predict the same class for each image.

Overall, despite our changes, our ViT's accuracy remained low. This could be due to a combination of factors, including the lack of translation invariance in the Transformer architecture and the size of our dataset. In contrast, ResNet is translation invariant due to its use of convolutional layers. Additionally, while ViT's have a better performance using large datasets compared to ResNet, Transformers typically underperform compared to ResNet models using small to medium datasets, since they require larger amounts of data to effectively learn relevant features. It is likely that the scale of our dataset limited the ViT's ability to extract meaningful data.

Data Cleaning

To ensure the quality and consistency of our dataset, we implemented several data cleaning steps across both the training and test datasets:

- **Folder Filtering**: Folders containing 30 or fewer images were removed. This threshold was set to maintain a balanced distribution of images per category.
- **Image Resizing with Aspect Ratio Preservation**: All images were resized to a consistent target size of 224 x 224 pixels. To avoid distortion, aspect ratios were preserved, and padding was added where necessary. This ensured that all images met the required input dimensions for the neural network without compromising image quality.
- **Format Standardization**: All images were converted to the JPEG format to establish consistent compatibility with the neural network and ensure consistency across the dataset.

Data Pre-Processing/Loading

The original Stanford Cars dataset evenly split images between test and train sets but did not include a validation set. To address this, we decided to adjust the distribution to a 80-10-10 split—80% for training, 10% for validation, and 10% for testing, also shifting the majority of images to our training set. We combined each car's test and train images into a unified set. From this merged set, we randomly divided the images into the new 80-10-10 distribution.

To improve our model's accuracy, we implemented the following data preprocessing steps:

- **Batching:** Our train, test, and validation datasets were divided in smaller batches of 32 images each, reducing memory usage and allowing faster training and faster convergence of our model [24].
- **Normalization of pixels:** The pixel values of each image were normalized by dividing each pixel value by 255, resulting in values within the range [0, 1]. This helped maintain stable gradients and improved computation speed [25].
- **Data prefetching:** Prefetching allowed the model to dynamically load the next batches based on current resources, minimizing wait time between batches. This allowed for faster training and optimized resource usage [26].
- **Data augmentation:** The training dataset was synthetically increased with data augmentation techniques, including random horizontal flips, and slight rotations and slight zooms to reduce overfitting [27].

Results and Discussion

CNN - Residual Neural Network

The ResNet model implemented here is inspired by Microsoft's 2015 ResNet architecture, leveraging residual blocks to enable deep learning without gradient vanishing or explosion issues. Each residual block in the model comprises two convolutional layers with batch normalization, connected by skip connections. These skip connections help preserve gradients across layers, allowing for more stable and effective training even in deep networks. The model processes input data through an initial convolution, followed by three residual blocks that increase the number of filters to capture complex features. A bottleneck 1x1 convolution aligns the dimensions between residual blocks to keep the feature map size consistent, which is essential for efficient learning without inflating computational costs. The model concludes with a global average pooling layer, flattening, and a dense softmax layer for classification. The use of sparse categorical cross-entropy as the loss function helps simplify

label handling, and the Adam optimizer accelerates convergence with minimal hyperparameter adjustments.

In our best result **Seventh Evaluation**, we observed the following from the **Loss Plot** and the **Accuracy Plot**:

The loss plot shows that the training loss steadily decreases, indicating that the model is effectively minimizing the loss on the training data. However, the validation loss initially decreases but exhibits fluctuations and a dramatic spike around epoch 40. This suggests potential overfitting, instability in the learning process, or a learning rate issue that caused the model to diverge temporarily on the validation data.

The accuracy plot reveals a growing gap between the training and validation accuracy. The training accuracy improves consistently and approaches 0.9, whereas the validation accuracy plateaus around 0.45 with fluctuations. This indicates overfitting, where the model is memorizing the training data but struggles to generalize to the validation set. The lack of improvement in validation accuracy suggests either insufficient model capacity, suboptimal hyperparameters, or inadequate regularization.

Initial Evaluation (0% Accuracy)
The first run, with 196 classes and no data augmentation, yielded 0% test accuracy (initally 1% at epoch 1, but decreased with each epoch) and no meaningful precision, recall, or F1 scores across classes. This was largely due to the high class count, which overwhelmed the model, and the computational burden from the 1x1 bottleneck convolution layer, which introduced inefficiencies. The confusion matrix revealed nearly random predictions, indicating severe underfitting. At this stage, the model lacked the ability to generalize effectively, likely due to insufficient gradient flow and difficulty learning distinctive features for each class.

Second Evaluation (13% Accuracy, Reduced Classes)
Reducing the class count from 196 to 48 improved the model's ability to distinguish between brands, resulting in a 13% accuracy increase. This reduction made the task more manageable, allowing the model to start learning specific brand features, though still at a limited level. Precision and recall scores showed modest improvement, particularly for brands like Aston Martin, Ferrari, and Chevrolet. The confusion matrix demonstrated an increase in correctly predicted classes but still displayed significant misclassifications, suggesting further optimization was needed.

Third Evaluation (17% Accuracy, Added Data Augmentation)
Adding data augmentation layers led to a 17% test accuracy, as augmentation introduced variability in the training set, allowing the model to generalize better across different lighting, contrast, and orientation conditions. Brands like Audi, BMW, Chrysler, and Ferrari saw further improvement in precision, recall, and F1 scores, indicating that the model was increasingly capable of recognizing brand-specific features. The confusion matrix showed increased clustering along the diagonal, especially for brands with a substantial number of samples, though performance on less-represented brands remained low.

Fourth Evaluation (32% Accuracy, 30 Epochs)
The model reached 32% accuracy after training for 30 epochs, allowing it to better internalize features across brands. This run saw notable improvements in both precision and recall for brands like Acura, Audi, BMW, Ferrari, and HUMMER. The longer training time allowed the model to recognize distinctive patterns, but brands with limited data still had poor performance, as reflected in the confusion matrix. This indicates a need for techniques to address class imbalance, such as weighted loss functions, to further improve generalization.

Fifth Evaluation (38% Accuracy, 50 Epochs, Overfitting Observed)
With 50 epochs, the model achieved a 38% test accuracy and an F1 score of 0.37. However, training accuracy reached 70%, suggesting overfitting as the validation accuracy stagnated at

36%. This was confirmed in the confusion matrix, where frequently seen brands showed strong performance while rarer brands remained problematic. The model learned specific details rather than general patterns, which compromised its ability to generalize. Regularization methods, such as dropout or weight decay, may help prevent this overfitting in future runs.

Sixth Evaluation (24% Accuracy, Extra Activation Layer)
Adding a separate activation layer on top of batch normalization within the residual blocks reduced test accuracy to 24% and F1 score to 0.2, indicating degraded model performance. The addition of an extra activation layer disrupted the learning dynamics within residual blocks, as seen in the lower scores across several brands. The confusion matrix shows fewer correct predictions than in the 32% model, particularly for brands previously performing well, like BMW and Ferrari. This outcome suggests that integrating activation functions directly within convolutional layers may provide more stable learning.

Seventh Evaluation (43% Accuracy, 60 Epochs, Overfitting Observed)
The model achieved 43% accuracy on the test set, with an F1 score of 0.41. While brands like HUMMER and Acura performed relatively well (F1 scores above 60%), rarer brands like Cadillac and Porsche had near-zero performance due to class imbalance. Overfitting was evident, with training accuracy reaching 91%, but validation accuracy lagging at 44.4%.

Eighth Evaluation (15% Accuracy, 62 Epochs, Hyperparameter Tuning)
The model achieved 15% accuracy on the test set, with an F1 score of 0.13. While hyperparameter tuning was applied with learning rate of 1e-3, the model's performance was relatively poor due to limited generalization capabilities. The large gap between training and testing accuracy suggests that further tuning or architectural changes are needed to improve the results.

Ninth Evaluation (27% Accuracy, 62 Epochs, Improved Hyperparameter Tuning)
The model achieved 27% accuracy on the test set, with an F1 score of 0.22. This marks a notable improvement over the eighth evaluation with learning rate of 1e-4, indicating that adjustments in hyperparameter tuning, such as learning rate and batch size, helped the model generalize better. However, the performance still requires significant optimization to handle the complexity of the classification task effectively.

Summary Table of Model Runs and Results

Evaluation	Model Changes	Test Accuracy	Precision	Recall	F1 Score	Observations
Initial (model_916520) Classification Report Confusion Matrix Heat Map	196 classes, no data augmentation	1%	0.00	0.00	0.00	Severe underfitting with random-like predictions; high computational burden from 1x1 convolution bottleneck
Second (model_916991) Classification Report	Reduced to 48 classes, 10 epochs, 10 steps/epoch	13%	0.11	0.13	0.07	Improved ability to distinguish brands; modest improvement for certain

Evaluation	Model Changes	Test Accuracy	Precision	Recall	F1 Score	Observations
Confusion Matrix Heat Map						brands like Aston Martin and Ferrari; some correct predictions appeared
Third (model_917357) Classification Report Confusion Matrix Heat Map	Added data augmentation	17%	0.14	0.17	0.13	Better generalization across varied lighting and contrast; increased performance for Audi, BMW, and Chrysler; improved diagonal clustering
Fourth (model_917372) Classification Report Confusion Matrix Heat Map Accuracy Plot Loss Plot	30 epochs full steps	32%	0.36	0.32	0.29	Significant performance gain; better feature extraction for major brands; less-represented brands still underperformed
Fifth (model_917799) Classification Report Confusion Matrix Heat Map Accuracy Plot Loss Plot	50 epochs full steps, observed overfitting	38%	0.47	0.38	0.37	Highest accuracy but clear overfitting with 70% train accuracy; frequent brands well-identified, less common brands poorly predicted
Sixth (model_926034) Classification Report	Extra activation layer after batch norm	24%	0.24	0.24	0.20	Performance declined with extra activation; reduced accuracy for most brands;

Evaluation	Model Changes	Test Accuracy	Precision	Recall	F1 Score	Observations
Confusion Matrix Heat Map Accuracy Plot Loss Plot						activation within convolution layers preferable for stability
Seventh (model_983557) Classification Report Confusion Matrix Heat Map Accuracy Plot Loss Plot	60 epochs full steps, observed overfitting	43%	0.46	0.43	0.41	Advanced data augmentation techniques such as random contrast and brightness adjustments and random cropping were added, which improved the model's accuracy by 10%.
Eighth (model_988968) Classification Report Confusion Matrix Heat Map Accuracy Plot Loss Plot	62 epochs full steps, initial hyperparameter tuning	15%	0.16	0.15	0.13	Initial hyperparameter tuning resulted in poor performance, highlighting the need for further optimization.
Ninth (model_989280) Classification Report Confusion Matrix Heat Map Accuracy Plot Loss Plot	62 epochs full steps, improved hyperparameter tuning	27%	0.22	0.27	0.22	Further hyperparameter adjustments led to a significant improvement in accuracy and generalization, indicating potential for optimization.

GMM - Gaussian Mixture Models

This approach performed significantly worse than our ResNet model, as our best results from our **Random Forest & SVM** only reached about 9% and 11% accuracy, respectively. We believe this occurred because our dataset was too small with too few examples for effective clustering. All the approaches used for this model only ever predicted a few brands.

Summary Table of Model Runs and Results

Evaluation	Model Changes	Test Accuracy	Precision	Recall	F1 Score
GMM vs K-Means (model_997376_100-iters_150-pca-components) Classification Report Confusion Matrix Heat Map Cluster Visualization Results for all runs done with this approach	Kmeans, K= 15	4%	0.04	0.04	0.04
DBSCAN and Spectral Clustering (model_998281_100-iters_50-pca-components) Classification Report Confusion Matrix Heat Map Cluster Visualization Results for all runs done with this approach	Kmeans K = 10	5%	0.03	0.05	0.04
Hybrid K-Means + GMM (model_998850_100-iters_50-pca-components) Classification Report Confusion Matrix Heat Map Cluster Visualization Results for all runs done with this approach	K = 10, covariance type = full	6%	0.02	0.06	0.03
Random Forest & SVM (model_999191_100-iters_100-pca-components) Classification Report	SVM	11%	0.01	0.11	0.02

Evaluation	Model Changes	Test Accuracy	Precision	Recall	F1 Score
Confusion Matrix Heat Map Results for all runs done with this approach					

ViT - Vision Transformer

ViT's performance was the worst among our three models, with the best run only achieving 8% accuracy. This is likely due to our small dataset and Transformers' lack of translation invariance compared to ResNet. Our first runs resulted in 1% accuracy, but we were able to slightly improve the model by changing the patch size and adjusting the sizes of our images.

However, for all runs, predictions were heavily concentrated on one brand of cars, indicating a consistent failure of the model to generalize. This brand also varied randomly across different runs. Runs also shared the issue of the training and test accuracy dramatically increasing or decreasing after a few epochs, further indicating instability of our model.

Effects of Resized Dataset

The second dataset we tested on, with reduced padding, resulted in conflicting results in the model's perfromance. Updating the dataset with resized images resulted in a slight increase of accuracy compared to the original condensed dataset for certain runs, while other runs also reflected a decrease in test accuracy over time.

Summary Table of Model Runs and Results

All confusion matrices reflect the concentrations of predictions for one class only.

Evaluation	Model Changes	Test Accuracy	Precision	Recall	F1 Score	Observations
model_1003948 Classification Report Confusion Matrix Heat Map Accuracy Plot Loss Plot	Separate depth layer, and data augmentation	1%	0.00	0.01	0.00	Test accuracy decreased over time, indicating that additional depth layer and data augmentation alone struggled to improve performance; all metrics near zero
Model 1004710 Classification Report Confusion	Run tanh activation in last MLP block	7%	0.00	0.07	0.01	Higher accuracy, with training accuracy increasing over time; test

Evaluation	Model Changes	Test Accuracy	Precision	Recall	F1 Score	Observations
Matrix Heat Map Accuracy Plot Loss Plot						accuracy and loss remains constant across epochs, indicating difficulty with generalizing features to unseen data
Model 1007534 Classification Report Confusion Matrix Heat Map Accuracy Plot Loss Plot	Augmentation, Tanh, drop-out, class weights (0.02)	2%	0.00	0.02	0.00	Accuracy declined after incorporating class weights; no improvement in any metrics compared to lack of usage of class weights
Model 1004965 Classification Report Confusion Matrix Heat Map Accuracy Plot	Resized condensed dataset, patch size 16 x 16	8%	0.01	0.08	0.01	Improvement in accuracy and highest precision among all runs; high rate of false positives, and low rate of actual positives, indicating failure to generalize remains

Next Steps

While basic augmentation is included, additional strategies could further diversify the training data and improve model robustness. Since we already have random rotations, cropping, random contrast and brightness, we could expand these augmentations by incorporating random hue and saturation adjustments or replace brightness/contrast adjustments with a full-color jittering implementation for more comprehensive augmentation.

For ResNet, increasing depth, using pretrained models like ResNet-50, and applying regularization methods (e.g., dropout, weight decay) can mitigate overfitting. Systematic hyperparameter tuning can further optimize performance.

The Vision Transformer (ViT) can benefit from pretrained models such as ViT-B/16 or DeiT and hybrid architectures combining CNNs with transformers. Addressing dataset noise, experimenting with patch sizes, and balancing classes through oversampling or class weights can also improve results.

A CNN-ViT hybrid model could combine the strengths of CNNs for local feature extraction and ViTs for capturing global relationships, enhancing the model's ability to classify cars with fine-grained differences. By using a pretrained CNN to extract features and feeding these as patches into a ViT, this approach could improve performance, especially with smaller datasets, while addressing the limitations of standalone CNNs or ViTs.

For GMM and clustering methods, integrating deep feature extraction from pretrained CNNs and refining the K-Means + GMM pipeline can enhance clustering. Exploring alternative methods like hierarchical clustering or mean-shift could yield better results.

Dataset expansion with additional car datasets like CompCars, along with balancing techniques like SMOTE, can reduce bias. Improved evaluation through k-fold cross-validation and interpretability tools (e.g., SHAP, Grad-CAM) can refine models. Finally, infrastructure optimizations like mixed-precision training, automated hyperparameter tuning, and deployment-focused pruning can ensure practical applications.

References

[1] S.S. Sarikan, A.M. Obayoglu, and O. Zilci, “Automated Vehicle Classification with Image Processing and Computational Intelligence,” Procedia Computer Science, vol. 114, pp. 515-522, 2017, doi: https://doi.org/10.1016/j.procs.2017.09.022.

[2] W. Maungmai and C. Nuthong, "Vehicle Classification with Deep Learning,"2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Singapore, 2019, pp. 294-298, doi: 10.1109/CCOMS.2019.8821689.

[3] Liang L, Ma H, Zhao L, Xie X, Hua C, Zhang M,and Zhang Y, “Vehicle Detection Algorithms for Autonomous Driving: A Review”, Sensors (Basel). Vol 10, May, pp. 13-24, 2024, doi: 10.3390/s24103088.

[4] M. Patel, “The Complete Guide to Image Preprocessing Techniques in Python,” Medium, Oct. 23, 2023. https://medium.com/@maahip1304/the-complete-guide-to-image-preprocessing-techniques-in-python-dca30804550c

[5] “11.8. Transformers for Vision — Dive into Deep Learning 1.0.3 documentation,” d2l.ai. https://d2l.ai/chapter_attention-mechanisms-and-transformers/vision-transformer.html

[6] A. A. M. Omer, “Image Classification Based on Vision Transformer,” Journal of Computer and Communications, vol. 12, no. 4, pp. 49-59, Apr. 2024, doi:

https://doi.org/10.4236/jcc.2024.124005.

[7] “What is a Convolution Neural Network?,” Hpe.com, 2023. https://www.hpe.com/us/en/what-is/convolutional-neural-network.html

[8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” arXiv.org, Dec. 10, 2015. https://arxiv.org/abs/1512.03385

[9] “(PDF) The Impact of Filter Size and Number of Filters on Classification Accuracy in CNN,” www.researchgate.net. https://www.researchgate.net/publication/342999107_The_Impact_of_Filter_Size_and_Number_of_Filters_on_Classification_Accuracy_in_CNN

[10] https://medium.com/swlh/how-to-create-a-residual-network-in-tensorflow-and-keras-cd97f6c62557

[11] https://github.com/Prakadeeswaran05/Customising-your-models-with-TensorFlow-2-Coursera/blob/main/Week%204%20Programming%20Assignment-Residual%20Network.ipynb

[12] https://www.youtube.com/watch?v=Q1JCrG1bJ-A

[13] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[14] https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

[15] https://www.geeksforgeeks.org/what-is-a-projection-layer-in-the-context-of-neural-networks/

[16] C. Bishop, “Pattern Recognition and Machine Learning,” Springer, 2006, pp. 352–356.

[17] H. W. Kuhn, “The Hungarian Method for the Assignment Problem,” Naval Research Logistics Quarterly, vol. 2, no. 1–2, pp. 83–97, 1955, doi: 10.1002/nav.3800020109.

[18] M. Ester, H. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” in Proc. Int. Conf. on Knowledge Discovery and Data Mining (KDD), 1996, pp. 226–231.

[19] P.S. Bradley and U.M. Fayyad, “Refining Initial Points for K-Means Clustering,” Proceedings of the Fifteenth International Conference on Machine Learning, San Francisco, CA, USA, pp. 91–99, 1998.

[20] J. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297, 1967.

[21] C.M. Bishop, “Pattern Recognition and Machine Learning,” Springer, 2006, pp. 423–457.
[22] A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” arXiv:2010.11929 [cs], Oct. 2020, Available: https://arxiv.org/abs/2010.11929
[23] Vision Transformer in Tensorflow, https://dzlab.github.io/notebooks/tensorflow/vision/classification/2021/10/01/vision_transformer.html (accessed Dec. 3, 2024).
[24] “Batch processing: The key to making your neural networks sing,” Tooliqa, https://www.tooli.qa/insights/batch-processing-the-key-to-making-your-neural-networks-sing (accessed Nov. 11, 2024).
[25] P. Harsh, “Normalization in image preprocessing: Scaling pixel values by 1/255,” Medium, https://medium.com/@patelharsh7458/normalization-in-image-preprocessing-scaling-pixel-values-by-1-255-111b2fa496d4 (accessed Nov. 11, 2024).
[26] “Better performance with the tf.data API,” TensorFlow, https://www.tensorflow.org/guide/data_performance (accessed Nov. 11, 2024).
[27] “Data augmentation,” TensorFlow, https://www.tensorflow.org/tutorials/images/data_augmentation (accessed Nov. 11, 2024).

Contributions

Name	Contributions
Madeline Liu Hou	**Report:** Intro, data-preprocessing/loading, Methods (ViT), and Results and Discussion (ViT). **Code:** data split/data augmentation, ViT model experiments and training
Sophia Isabel Marples Rodriguez	**Report:** Problem Definition, Results and Discussion (GMM/ViT), Next Steps
Jiya Varma	**Report:** Methods (ResNet, ViT). **Code:** ResNet model implementation and training. ViT model implementation, experiments, and training
Nicole Hernandez Canales	**Report:** Methods (GMM), Results and Discussion (ResNet: runs 1-4), and Next Steps.

Name	Contributions
	Code: all files under the GMM directory in the project Github
Mengzhu Ou	**Report:** Results and Discussion (ResNet: runs 5-9), Demo Video Editing **Code:** Data Augmentation Improvement, Accuracy Plot and Loss Plot generation