Project Report: Vehicle Classification with ResNet

1. Introduction:

The project focuses on the development of a vehicle classification system leveraging the ResNet50 architecture. Vehicle classification plays a pivotal role in diverse applications such as traffic management, surveillance, and autonomous driving. The goal is to employ deep learning techniques to accurately identify and categorize vehicles from images, providing a foundation for more advanced applications in the realm of computer vision. The utilization of the ResNet model, known for its effectiveness in image recognition tasks, ensures robust performance and allows for insightful visualizations to understand the model's decision-making process. This report will detail the key components of the project, from data loading and model training to evaluation metrics and visualizations, culminating in a comprehensive analysis of the vehicle classification system

2. Data Loading and Preprocessing:

- The project begins with the preparation of the dataset for training and evaluation. The vehicle images are organized into separate directories for training and testing, and PyTorch's ImageFolder class is employed for efficient loading. To enhance the model's robustness, data augmentation techniques are applied during training. Specifically, random resized cropping and horizontal flipping are utilized to augment the training set, ensuring that the model generalizes well to diverse scenarios.
- Two distinct transformations, train_transform and test_transform, are defined to preprocess the images for the training and testing phases. These transformations include resizing, cropping, and normalization to align the data with the requirements of the ResNet model. The training and testing datasets are then loaded using PyTorch's DataLoader to facilitate efficient batch processing during model training and evaluation.
- The dataset is structured in a way that allows for a seamless integration with the ResNet model, enabling the training of a robust vehicle classification system. The subsequent sections of the report will delve into the model architecture, training process, and evaluation metrics.

3. Model Definition and Training:

- The core of the project revolves around the implementation and training of a ResNet-based neural network for vehicle classification. The ResNet architecture,

specifically ResNet-50, is utilized as a feature extractor due to its proven effectiveness in image classification tasks.

- The model definition is encapsulated within the ResNetModel class, which inherits from PyTorch's nn.Module. This class initializes a pre-trained ResNet-50 model and adapts its last fully connected layer to accommodate the specific number of classes in the vehicle classification task. The number of classes is determined dynamically based on the training dataset.
- To tailor the model for the vehicle classification task, the last fully connected layer's output dimensions are modified to match the number of classes. The initial layers of the ResNet model are kept frozen (non-trainable) to retain the pre-trained features, while the last fully connected layer is marked as trainable.
- The training process involves the use of a cross-entropy loss function and the Adam optimizer. A learning rate scheduler is incorporated to adjust the learning rate during training, enhancing convergence. The training loop iterates through the specified number of epochs, and for each epoch, the model is trained on batches of data from the training set.
- During training, the model's performance is monitored through the display of the training loss for each epoch. The learning rate is adjusted using the scheduler to optimize convergence. The trained model is then ready for evaluation on the test dataset.
- The subsequent sections of the report will delve into the model evaluation, including the calculation of performance metrics, visualization of filters and feature maps, and the generation of a comprehensive classification report.

4. Evaluation and Metrics Calculation:

The evaluation phase of the project focuses on assessing the performance of the trained ResNet model on the test dataset. Various metrics are employed to provide a comprehensive understanding of the model's effectiveness in vehicle classification. The first step involves utilizing a dedicated function, get_predictions(loader), to generate predictions and probabilities for each image in the test dataset. This function ensures that the model and inputs are appropriately moved to the available device, whether it be GPU or CPU. The predictions are then compared against the true labels to calculate key performance metrics.

Metrics calculated include:

ROC AUC Score: The Receiver Operating Characteristic Area Under the Curve (ROC AUC) score is computed to gauge the model's ability to distinguish between classes. The multi-class 'one-vs-rest' approach is adopted for this calculation.

Confusion Matrix: A confusion matrix is generated to provide a detailed breakdown of the model's predictions. It showcases the number of true positive, true negative, false positive, and false negative instances for each class.
Accuracy: The overall accuracy of the model is determined by comparing the predicted labels with the true labels and calculating the ratio of correctly classified samples.
Classification Report: A comprehensive classification report is generated, presenting precision, recall, F1-score, and support for each class. This report offers insights into the model's performance on a per-class basis.
Additionally, the probabilities predicted by the model are modified and normalized to ensure consistency and reliability. This involves extracting only the probabilities corresponding to the classes present in the dataset and normalizing them to sum up to 1.0 over classes.
The calculated metrics and visualizations contribute to a thorough understanding of the model's strengths and areas for improvement in the vehicle classification task. The ensuing sections of the report will explore the visualization of filters, feature maps, and provide examples of model predictions on test images.

5. Model Visualization:

To gain insights into the inner workings of the ResNet model, visualizations are conducted to showcase filters in specific layers and feature maps generated during the inference process.
Filter Visualization: The initial convolution block's filters are visualized, providing a glimpse into the patterns and features the model learns at its early stages. Filters in subsequent layers, specifically in the layer4 block, are also examined to understand the hierarchical feature extraction.
Feature Map Visualization: Feature maps for the initial convolution block are generated by applying the model to example images from each class in the training dataset. These feature maps illustrate the regions of interest and activations detected by the model, shedding light on its decision-making process.
These visualizations aid in interpreting the ResNet model's learning patterns and understanding which features contribute to its predictions. They provide valuable insights for fine-tuning and optimization, allowing for a more informed approach to further model development and refinement.

6. Image Prediction and Display:

In this section, the trained ResNet model is utilized to perform predictions on individual test images. The process involves loading an image, preprocessing it to align with the model's input requirements, and then utilizing the model to obtain predictions. The

predicted label is compared with the true label, and the result is displayed visually for better comprehension.
Image Loading and Preprocessing: A test image of a vehicle is loaded and converted into a PyTorch tensor. The necessary transformations, such as normalization, are applied to match the preprocessing steps during model training.
Model Inference: The preprocessed image is fed into the trained ResNet model to obtain predictions. The predicted label is then compared with the true label for evaluation.
Visualization: The original image is denormalized and displayed alongside information about the true and predicted labels. This visual representation provides a qualitative assessment of the model's performance on individual test samples.
This section aims to demonstrate the practical application of the trained model on real-world images and offers a visual confirmation of its classification accuracy.

7. Model Saving:

After the model is successfully trained on the vehicle classification task, it is essential to save the trained weights for future use and deployment. In this section, the PyTorch torch.save() function is employed to store the model's state dictionary, which includes all the learnable parameters.
State Dictionary: The state dictionary contains the learned weights and biases of the model, allowing it to be reconstructed later for inference or further training.
File Format: The model is saved in the PyTorch-specific format with a .pth extension, preserving the hierarchical structure of the model.
Purpose: Saving the model serves various purposes, such as sharing the trained model with others, deploying it in production environments, or resuming training from a specific checkpoint.
By preserving the model state, this step ensures that the knowledge gained during training is encapsulated and can be leveraged for diverse applications beyond the initial training phase.

8. Comments and Suggestions:

Throughout the implementation of the vehicle classification project using ResNet, several aspects have been considered to enhance code readability, maintainability, and overall project success. Below are some comments and suggestions for further improvement:
Code Documentation: While the code contains comments explaining individual sections, consider expanding documentation to include more details about complex

functions, critical decision points, and the overall structure of the project. This can greatly aid anyone reviewing or collaborating on the code.
Modularization: The code could benefit from further modularization. Breaking down the functionalities into separate modules or functions enhances code readability, promotes reusability, and simplifies debugging. This modular approach can also facilitate the testing of individual components.
Parameterization: Hardcoded values, such as file paths and hyperparameters, are present in the code. Consider parameterizing these values, making the code more flexible and adaptable to changes in the dataset, model architecture, or training parameters.
Logging: Implementing a logging system can help track the training progress, record key metrics, and capture any issues that may arise during execution. This provides a more comprehensive view of the training process.
Data Exploration: Although not explicitly included in the provided code, it's valuable to conduct data exploration and analysis before training. Understanding the dataset's characteristics can influence data preprocessing decisions and contribute to the overall success of the model.
Code Validation: Implementing code validation, such as type checking and input validation, adds an extra layer of robustness to the code. This ensures that the code is less prone to errors and can handle unexpected situations gracefully.
Testing: Incorporating unit tests or integration tests is crucial for verifying that each component of the code functions as intended. Testing helps catch bugs early in development and ensures the reliability of the entire system.
User Interaction: If applicable, consider incorporating user interaction features, such as command-line arguments or a graphical user interface, to enhance the user experience and allow for easy customization of the model and its parameters.

9. Conclusion:
In summary, the ResNet-based vehicle classification project excelled in accurately identifying diverse vehicle images. The customized ResNet50, trained with meticulous preprocessing and optimization, showcased robust performance. Evaluation metrics provided a comprehensive overview, and visualizations offered insights into the model's inner workings. The project's practicality was demonstrated through image predictions, and the saved model ensures future applicability and potential enhancements in image classification and computer vision.