Car Classification for Surveillance Systems and Automated Traffic Monitoring

A PROJECT REPORT

Submitted by

SANJU K – 21MID0108

AJAY SAAM – 21MID0158

DHANUSH S - 21MID0183

THARUN J – 21MID0221

CS13001 Soft Computing Techniques

Project Guide

Dr. VIJAYASHERLY V

Associate Professor Grade 1

School of Computer Science and Engineering

MTECH(Integrated) COMPUTER SCIENCE AND TECHNOLOGY WITH SPECILIZATION DATA SCIENCE

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Fall Semester 2024-2025

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	**Full Form**
CNN	Convolutional Neural Network
ITS	Intelligent Transportation System
ReLU	Rectified Linear Unit
GPU	Graphics Processing Unit
RNN	Recurrent Neural Network
LIDAR	Light Detection and Ranging

SVM	Support Vector Machine
HOG	Histogram of Oriented Gradients
FF-CMNET	Feature Fusion Correlation Mapping Network
IT	Information Technology

Abstract

The rapid evolution of deep learning and computer vision technologies has opened new avenues for intelligent traffic systems, particularly in vehicle identification and classification. This project centers on employing Convolutional Neural Networks (CNNs) to develop an advanced car classification system, tailored for integration with surveillance infrastructure and automated traffic monitoring. Utilizing a well-structured dataset comprising images of seven distinct car categories—Audi, Mahindra Scorpio, Hyundai Creta, Rolls Royce, Tata Safari, Maruti Suzuki Swift, and Toyota Innova—the system is designed to accurately identify vehicles under diverse conditions. The model leverages image preprocessing, data augmentation, and optimized CNN layers to achieve robust classification performance. Experimental results demonstrate a classification accuracy of approximately 98%, confirming the model's effectiveness and adaptability for real-time traffic surveillance and law enforcement applications. This work contributes to the growing domain of intelligent transportation by offering a scalable, efficient, and high-performing vehicle classification framework.

Chapter 1

Introduction

In the context of urbanization and rapidly increasing vehicle populations, the need for intelligent and automated traffic management systems has become more pressing than ever. Intelligent Transportation Systems (ITS) aim to address these challenges through technologies that enable efficient traffic flow, reduced congestion, and enhanced public safety. A core component of ITS is the ability to accurately identify and classify vehicles from real-time video or image input, enabling use cases such as automatic toll collection, traffic rule enforcement, intelligent parking systems, and tracking of suspicious vehicles.

Traditional vehicle classification methods have relied on rule-based image processing techniques or simple machine learning models. However, these approaches often struggle with real-world complexities such as varying lighting conditions, occlusions, different viewing angles, and noisy backgrounds. To overcome these limitations, this study investigates the use of Convolutional Neural Networks (CNNs), a class of deep learning models known for their exceptional performance in image classification tasks.

By leveraging CNNs, this research aims to build a scalable and accurate car classification model that can function reliably in surveillance environments and traffic monitoring setups.

This system is particularly relevant for enhancing the operational efficiency of city traffic departments, law enforcement agencies, and infrastructure authorities.

Aim

The primary aim of this project is to design and implement a deep learning-based car classification system using Convolutional Neural Networks (CNNs) that can:

- **Accurately classify seven specific car models** commonly found in the Indian subcontinent.
- **Integrate seamlessly with surveillance and traffic monitoring systems** for real-time identification.
- **Achieve a target accuracy of approximately 98%**, ensuring both practical usability and high reliability.

This aim supports the broader goal of building smarter urban infrastructure through the application of advanced AI technologies.

Objective

To realize the above aim, the project pursues the following specific objectives:

1. **Dataset Acquisition and Preparation:** Utilize a labeled dataset from Kaggle that includes over 4,000 images across seven car categories. Apply preprocessing techniques such as image resizing, normalization, and data augmentation to enhance model robustness.
2. **CNN Model Development:** Construct a CNN architecture consisting of convolutional, pooling, and dense layers with ReLU activations and dropout to prevent overfitting. Ensure the model captures hierarchical features essential for car classification.
3. **Model Training and Optimization:** Train the CNN on the preprocessed dataset using appropriate loss functions and optimizers (e.g., categorical crossentropy and Adam). Perform hyperparameter tuning to improve model accuracy and reduce loss.
4. **Performance Evaluation:** Evaluate the model using validation and test datasets. Measure key metrics such as accuracy, loss, precision, recall, and F1-score. Analyze performance trends across epochs to ensure effective learning.
5. **Real-World Testing and Deployment:** Test the model's ability to generalize by applying it to real-world car images. Explore options for deployment in a practical surveillance system or as part of a traffic analytics platform.

Motivation

The growing density of vehicular traffic, particularly in developing urban regions, poses significant challenges for traffic regulation, congestion management, and road safety. Manual monitoring methods are inefficient and prone to human error, especially in high-volume areas. The integration of artificial intelligence into surveillance systems presents a scalable solution for automating traffic analysis and vehicle tracking.

Moreover, accurately identifying vehicle make and model in real-time opens up a range of advanced applications—from automated toll booths to intelligent alert systems for stolen or suspicious vehicles. CNNs have revolutionized computer vision and are particularly adept at identifying intricate visual patterns, making them ideal for vehicle classification tasks.

This project is motivated by the opportunity to bridge the gap between theoretical deep learning advancements and their practical implementation in intelligent transportation systems. The societal and infrastructural benefits of such a solution, including enhanced traffic safety, efficient law enforcement, and reduced operational costs, provide strong incentives for this research.

Problem Statement

The core problem addressed in this project is:

"How can deep learning, specifically Convolutional Neural Networks, be utilized to build a reliable and high-accuracy vehicle classification system capable of identifying multiple car types from image input in real-time, under real-world surveillance conditions?"

II. Literature Survey

A. Survey of Existing Models and Works

Several deep learning models have been proposed for vehicle classification, each aiming to improve performance through architectural innovations or dataset contributions. Saputra et al. utilized DenseNet and convolutional neural networks (CNNs) for multi-class vehicle detection, achieving high accuracy due to efficient feature reuse and dense connections between layers [1]. Their approach demonstrated the effectiveness of deeper architectures in capturing inter-class variations among vehicle types such as cars, buses, and trucks.

Chen et al. employed 3D spatial data in conjunction with deep networks for part-wise vehicle classification [2]. By incorporating depth cues and structural representations, their method improved the differentiation of visually similar vehicle types and offered enhanced performance in fine-grained classification tasks.

Yu and Jin proposed FFCMNET, a specialized CNN designed for fine-grained vehicle classification [3]. The model introduced a feature correlation mapping mechanism that improved intra-class feature representation. This approach significantly enhanced performance on challenging datasets where vehicles had subtle differences.

Syafei conducted a comparative analysis of MobileNet and ResNet architectures [4]. MobileNet, being lightweight and computationally efficient, showed better suitability for mobile applications, whereas ResNet provided superior accuracy at the cost of higher computational complexity. This study highlighted the trade-offs between model size, accuracy, and inference time.

Kuhn introduced the BRCars dataset, a large-scale and diverse vehicle image collection designed to facilitate robust model training and evaluation under real-world conditions [5]. The dataset includes variations in lighting, viewpoints, and background environments, which are critical for developing generalizable models.

B. Summary, Gaps, Limitations, and Future Work

Although the reviewed models achieve promising accuracy, certain limitations remain with regard to real-world deployment and adaptability. A primary concern is the lack of consideration for Indian traffic scenarios, which involve a heterogeneous mix of vehicles such as two-wheelers, auto-rickshaws, and commercial vehicles, alongside complex and dense road conditions.

Moreover, many existing datasets are imbalanced and lack sufficient diversity in terms of viewpoints, occlusion, weather, and lighting. This often results in overfitting and reduced generalization capability when models are exposed to real-world inputs. Additionally, real-time performance metrics such as inference latency and computational load are frequently underreported, making it difficult to evaluate model efficiency for embedded or mobile deployment.

Future work should prioritize the development of lightweight models through pruning, quantization, and knowledge distillation to enable real-time inference on edge devices. Expanding classification capabilities to include additional vehicle categories, particularly two-wheelers and non-standard commercial vehicles, is also essential. Furthermore, integrating multi-modal data such as video streams, LIDAR, and thermal imaging could enhance classification robustness. Finally, domain adaptation techniques should be employed to better align training data with deployment environments, especially in diverse and dynamic traffic settings like those in India.

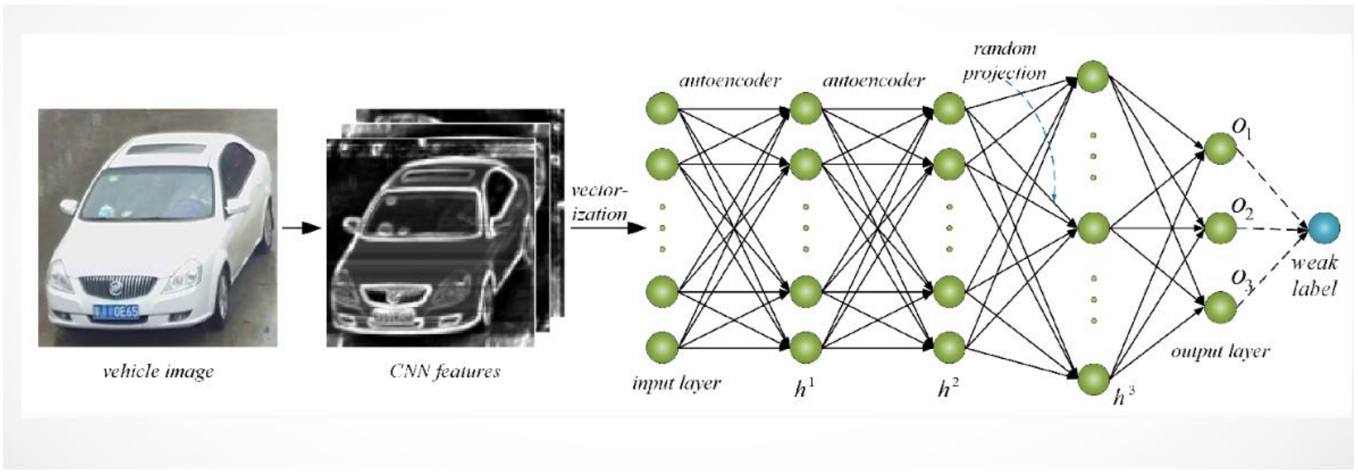


Fig 2.1

3. Overview of the Proposed System

3.1 Introduction

This section presents a detailed overview of the proposed system designed for classifying vehicles using deep learning techniques. The system uses a Convolutional Neural Network (CNN) trained on a specially curated dataset of car images, focusing particularly on Indian car models. These images are classified into seven distinct categories based on visual characteristics such as shape, size, and design.

The system is designed in a modular way, consisting of several components: data acquisition, data preprocessing, model architecture, training and validation, testing and misclassification analysis, and real-time deployment. Each module plays a key role in building a system that is both accurate and efficient for real-world use.

3.2 Framework and Architecture

The system architecture follows a structured flow, where each stage contributes to preparing the data, building the model, and deploying it effectively.

1. Data Acquisition

A diverse dataset of car images was created by collecting data from open-source datasets and web sources. Care was taken to include various car types that are common in Indian roads. The images reflect different lighting conditions, angles, and environments to ensure the model can generalize well to real-world scenarios.

2. Data Preprocessing

Preprocessing includes resizing all images to a standard size, normalizing the pixel values, and applying image augmentation techniques. These techniques include rotation, flipping, zooming, contrast adjustment, and cropping. The goal is to expand the dataset artificially and help the model learn features that are consistent under various conditions.

3. Model Architecture (CNN)

The CNN model is custom-built specifically for this task. Unlike approaches that use pre-trained models, this architecture is designed from scratch. It includes:

- Two convolutional layers for feature extraction.
- Max pooling layers to reduce spatial dimensions.
- Dropout layers to prevent overfitting.
- Dense (fully connected) layers to interpret the features.
- A final output layer with softmax activation for multi-class classification.

4. Training and Validation

The model is trained using labeled data, with a common split of 80% for training and 20% for validation. The loss function used is categorical cross-entropy, and the optimizer used is Adam. During training, validation metrics are monitored to prevent overfitting and ensure good generalization to unseen data.

5. Testing and Misclassification Analysis

After training, the model is evaluated on a test set. Performance is measured using accuracy, precision, recall, and confusion matrix. Misclassified images are studied to understand patterns of failure—such as images taken in poor lighting or those with occluded vehicles—so the system can be improved further.

6. Real-Time Deployment

The trained model is deployed on a web interface that allows users to upload an image and get real-time predictions. The model has been optimized for speed so that it can run on standard devices without the need for specialized hardware like GPUs.

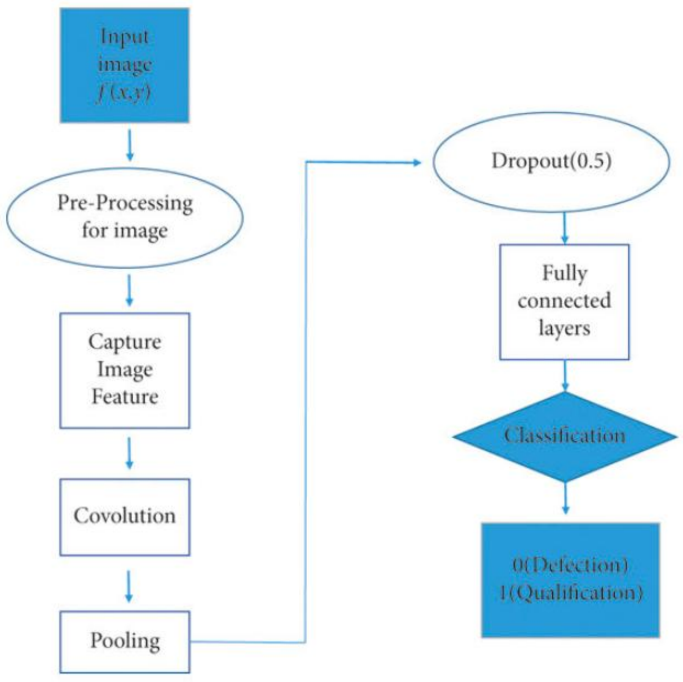


Fig 3.2

3.2.1 Uniqueness and Novelty

What makes this system unique is that the CNN model is built entirely from scratch, specifically tuned for Indian traffic conditions. Unlike generic models that use architectures like ResNet or VGG, this model is lighter and more focused.

Another novel feature is the testing approach. The model is not only evaluated on a structured dataset but also on images collected from the web. These images often include poor lighting, blurriness, or complex backgrounds, making them excellent tests for real-world performance. Additionally, heavy use of data augmentation and dropout ensures the model is robust and less prone to overfitting.

3.3 Proposed System Model

The CNN performs three major operations:

1. Convolution:

Each image is passed through a filter that scans it and applies a mathematical operation to extract features. The result is then passed through an activation function called ReLU. This can be written as:

C = ReLU(W * I + b)

Where:

- C is the resulting feature map
- W is the weight matrix (filter)
- I is the input image
- b is the bias
    -
        - indicates convolution

2. Pooling:

After convolution, max pooling is used to reduce the size of the feature map while keeping the most important information. It can be expressed as:

P = max_pool(C)

Where P is the pooled feature map and C is the input from the convolution layer.

3. Fully Connected Layers and Softmax:

After flattening the feature maps, the fully connected layers make the final decision. The last layer uses softmax to give probabilities for each class:

y_i = exp(z_i) / sum(exp(z_j)) for j = 1 to 7

Where:

- y_i is the probability of class i

- z_i is the input score for class i
- exp refers to the exponential function
- The denominator is the sum of exponentials of all class scores (from 1 to 7)

The class with the highest probability is chosen as the final prediction.

3.3.1 Block Diagram

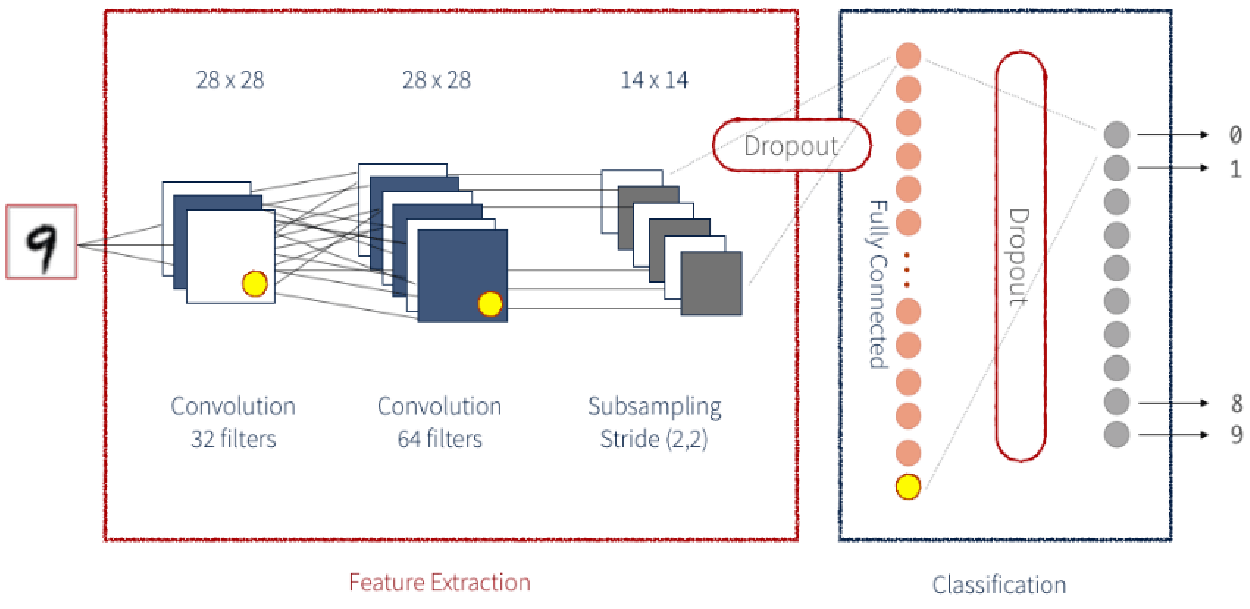


Fig 3.3

---

4. Proposed System Analysis and Design

4.1 Overview of the CNN Architecture

The core of the proposed system is a Convolutional Neural Network (CNN) designed to perform vehicle classification based on images. CNNs are particularly well-suited for image processing tasks because they are capable of automatically learning spatial hierarchies of features, which are essential in object recognition tasks like vehicle classification. The proposed CNN is designed to strike a balance between model complexity and computational efficiency, with the ultimate goal of producing high classification accuracy while minimizing computational load.

The architecture of the CNN is designed with several critical components:

1. **Convolutional Layers**: These layers are responsible for extracting local features from the input image. Convolutional layers utilize filters or kernels that move over the image and apply mathematical operations to highlight different features, such as edges, textures, and shapes.

2. **Pooling Layers**: These layers perform downsampling of the feature maps produced by the convolutional layers. Pooling reduces the spatial dimensions of the feature maps, which helps reduce computational complexity and mitigate the risk of overfitting.
3. **Dropout Layers**: Dropout is a regularization technique used to prevent overfitting. During training, dropout randomly disables a fraction of the neurons, forcing the network to learn more robust features and not become reliant on specific neurons.
4. **Dense Layers**: After the convolutional and pooling layers have extracted the relevant features, dense layers are used to perform the final decision-making process. These layers are fully connected layers that integrate the learned features to classify the image into one of the predefined classes.
5. **Softmax Activation**: The final layer uses a softmax activation function to compute the probability distribution of the predicted classes, allowing the model to choose the most likely class for a given input.

4.2 Layer-Wise Analysis

4.2.1 Convolutional Layers

The convolutional layers are the foundational building blocks of the CNN. They are responsible for the automatic extraction of features from the input image. Convolutional operations allow the network to learn local patterns, such as edges, corners, and textures, which are essential for recognizing vehicles. These patterns are learned through filters (or kernels) that are applied to the input image.

For each convolution operation, the filter is moved across the image, and at each position, a dot product between the filter and the portion of the image it covers is computed. This process is known as the **convolution operation** and can be expressed as:

$$\mathbf{C = ReLU(W * I + b)}$$

Where:

- **C** is the resulting feature map.
- **W** is the filter matrix (weights of the kernel).
- **I** is the input image or the feature map from the previous layer.
- **b** is the bias term.
- **ReLU** is the activation function applied to introduce non-linearity.

The filters are learned during the training process, allowing the network to adapt to the specific features relevant to the vehicle classification task. By stacking multiple convolutional layers, the network can learn increasingly abstract features, starting from simple edges and textures to more complex features such as vehicle shapes and body parts.

4.2.2 Pooling Layers

Pooling layers are used to reduce the spatial dimensions of the feature maps while retaining the most important information. This operation reduces the computational load, speeds up training, and helps prevent overfitting by reducing the number of parameters in the model. The most common pooling operation is **max pooling**, which selects the maximum value from a set of pixels within a fixed-size window.

Mathematically, the pooling operation can be written as:

P = max_pool(C)

Where:

- **P** is the pooled feature map.
- **C** is the input feature map from the convolutional layer.
- **max_pool** is the max pooling operation applied across a defined window.

By performing max pooling, the network captures the most important features while discarding less relevant information. This results in a more compact representation of the input image, which reduces the complexity of the model and helps prevent overfitting.

4.2.3 Dropout Layers

Dropout is a regularization technique that is used to prevent overfitting in neural networks. During training, dropout randomly deactivates a subset of neurons, forcing the network to rely on multiple neurons rather than memorizing specific ones. This promotes the learning of more generalized features.

Dropout is applied after the convolutional and pooling layers to ensure that the network does not become too specialized to the training data. The fraction of neurons to be deactivated is typically a hyperparameter that is tuned during model training. A common value is 0.5, which means that 50% of the neurons are randomly deactivated during each training step.

The mathematical representation of the dropout operation is:

y = f(x) * r

Where:

- **y** is the output of the neuron after dropout.
- **f(x)** is the output of the neuron before applying dropout.
- **r** is a random variable that determines whether the neuron is kept (1) or dropped out (0).

By applying dropout, the model becomes less sensitive to specific neurons, and instead, it learns to use a broader set of features, leading to better generalization.

4.2.4 Dense Layers

Dense layers are used to integrate the high-level features extracted by the convolutional and pooling layers and make the final decision about the class of the input image. A dense layer is a fully connected layer, meaning that each neuron is connected to every neuron in the previous layer. The output of a dense layer is computed as a weighted sum of the inputs, followed by an activation function.

Mathematically, a dense layer can be expressed as:

y = W * x + b

Where:

- **y** is the output of the dense layer.
- **W** is the weight matrix.
- **x** is the input to the dense layer (from the previous layer).
- **b** is the bias term.

After passing through the dense layers, the network produces a set of values that are input to the final softmax layer.

4.2.5 Softmax Activation

The final layer of the CNN is a softmax layer, which is used to convert the raw output values from the dense layers into a probability distribution over the classes. The softmax function ensures that the outputs sum to one, making them interpretable as probabilities.

The softmax function can be expressed as:

y_i = exp(z_i) / sum(exp(z_j)) for j = 1 to K

Where:

- **y_i** is the probability of class **i**.
- **z_i** is the score (logit) for class **i**.
- **K** is the number of classes (in this case, 7 vehicle classes).

The class with the highest probability is selected as the final prediction.

4.3 Design Considerations

4.3.1 Reduced Parameter Count

One of the key design goals of the proposed CNN architecture was to minimize the parameter count while maintaining the learning capacity of the network. By using smaller convolutional filters, applying pooling layers, and introducing dropout, the model significantly reduces the number of parameters compared to traditional fully connected networks. This not only makes the network more computationally efficient but also helps prevent overfitting.

4.3.2 Preservation of Learning Capacity

While the parameter count is reduced, the design ensures that the network retains its capacity to learn complex features. The use of multiple convolutional layers allows the network to learn hierarchical features, with each layer learning increasingly abstract representations. Pooling layers ensure that the network focuses on the most important features, while dropout ensures that the model generalizes well.

4.3.3 Real-Time Inference

Another important design consideration was ensuring that the model could perform real-time inference. This was achieved by optimizing the network architecture, using fewer layers and smaller filters where appropriate, to reduce computational time. The model is lightweight and designed to run on standard hardware, making it suitable for deployment in real-time vehicle classification applications.

4.4 Summary of System Design

In summary, the CNN architecture is designed with several key features in mind: it efficiently extracts features, reduces computational complexity, and prevents overfitting. The modular design, with convolutional layers, pooling, dropout, and dense layers, allows the model to learn from both low-level and high-level features of the input images. The softmax layer at the end ensures that the network produces a reliable probability distribution, making it possible to classify images accurately. The careful balance between reducing parameter count and maintaining learning capacity makes this system highly efficient for vehicle classification in real-time scenarios.

---

5. Implementation

The implementation of the vehicle classification system is carried out using Python, leveraging the Keras deep learning library with the TensorFlow backend. Keras provides a high-level interface for building and training deep learning models, which significantly simplifies the process of defining neural network architectures and optimizing them for real-world tasks. The primary goal of the implementation is to design a CNN model capable of classifying car images into seven different categories based on visual features.

5.1 Data Preparation

The dataset used for training, validation, and testing the model consists of a total of 4,165 images. These images are curated from various sources, ensuring that they represent a wide range of vehicles and environmental conditions. The dataset is split into three distinct subsets:

- **Training Set**: 80% of the images are used for training the model. This set is used to adjust the model's weights and biases during the learning process.

- **Validation Set**: 10% of the images are used for validating the model during training. This set helps monitor the model's performance on unseen data and prevents overfitting.
- **Testing Set**: The remaining 10% is used for evaluating the model's final performance after training is complete.

The images are preprocessed by resizing them to a uniform size (e.g., 224x224 pixels), normalizing the pixel values to a range of [0, 1], and performing data augmentation techniques such as random rotations, zooms, and horizontal flips. These steps enhance the model’s ability to generalize across varying conditions and prevent overfitting.

5.2 CNN Architecture Design

The CNN architecture is designed to perform multi-class vehicle classification based on the preprocessed images. The model is composed of several layers that are crucial for feature extraction, dimensionality reduction, regularization, and final classification.

1. Convolutional Layer (Conv2D)

The first layer of the CNN is a convolutional layer with 32 filters (kernels), each of size 3x3. This layer is responsible for automatically learning low-level features from the input image, such as edges, textures, and basic shapes. Each filter moves across the image and applies a convolution operation to detect these features. The number of filters in this layer determines the depth of the output feature maps, and the size of the filter affects the size of the features learned by the network.

- **Layer Definition**:

  Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3))

- **Explanation**:
  - **32 filters**: This means the layer will learn 32 distinct features in the input image. Each filter is applied to the image in the convolution operation.
  - **3x3 kernel**: The size of the filter. A 3x3 filter is commonly used as it allows the network to learn small, local patterns in the image.
  - **ReLU activation**: The Rectified Linear Unit (ReLU) is used as the activation function, introducing non-linearity into the model to help it learn complex patterns.
  - **Input shape**: The input shape is defined as (224, 224, 3), where 224x224 represents the dimensions of the image and 3 is the number of color channels (RGB).

2. Pooling Layer (MaxPooling2D)

Following the convolutional layer, the next layer is a max pooling layer. This layer performs downsampling on the feature map produced by the convolutional layer. By selecting the maximum value in a pool size (typically 2x2), the pooling layer reduces the spatial dimensions of the feature maps, which helps reduce computational complexity and memory usage.

- **Layer Definition**:

  ```
  MaxPooling2D(pool_size=(2, 2))
  ```

- **Explanation**:
  - **Pool size (2, 2)**: This means the pooling layer takes 2x2 blocks from the feature map and returns the maximum value from each block.
  - **Downsampling**: The pooling layer reduces the feature map dimensions by a factor of 2, which also reduces the number of parameters in subsequent layers.

3. Dense Layers

After several convolutional and pooling layers, the CNN model transitions into fully connected layers (also known as dense layers). These layers integrate all of the learned features and make the final decision regarding which vehicle class the image belongs to.

- The first dense layer has 96 units, followed by a second dense layer with 32 units. These layers are designed to combine the learned features from the previous layers and perform the final classification.
- **Layer Definitions**:

  ```
  Dense(96, activation='relu')
  Dense(32, activation='relu')
  ```

- **Explanation**:
  - **96 and 32 units**: These values represent the number of neurons in each dense layer. The first dense layer has more neurons to capture complex relationships between features, while the second layer reduces the dimensionality.
  - **ReLU activation**: ReLU is again used in the dense layers to introduce non-linearity, enabling the model to learn complex patterns in the data.

4. Dropout Layer

To prevent overfitting, a dropout layer with a rate of 0.4 is added after the dense layers. This means that 40% of the neurons will be randomly dropped during training, forcing the model to generalize better by preventing reliance on specific neurons.

- **Layer Definition**:

  ```
  Dropout(0.4)
  ```

- **Explanation**:
  - **Dropout rate (0.4)**: This means that 40% of the neurons in this layer will be randomly "turned off" during training. This helps to prevent the model from overfitting to the training data and encourages the network to develop more robust features.

5. Output Layer

The final layer of the CNN is the output layer, which has 7 units corresponding to the seven vehicle classes. A **softmax activation function** is applied to produce a probability distribution over the classes, where the class with the highest probability is selected as the model’s prediction.

- **Layer Definition**:

  Dense(7, activation='softmax')

- **Explanation**:
  - **7 units**: The output layer consists of 7 neurons, each representing one of the vehicle classes.
  - **Softmax activation**: The softmax function ensures that the output values are probabilities that sum to 1. It converts the raw scores (logits) from the dense layer into probabilities, making it easy to determine which class has the highest likelihood.

5.3 Model Compilation and Training

The model is compiled with the following settings:

- **Loss Function**: Categorical Crossentropy, which is used for multi-class classification tasks. This loss function compares the predicted class probabilities to the true class labels and computes the loss accordingly.
- **Optimizer**: Adam optimizer, which is an adaptive learning rate optimization algorithm. Adam adjusts the learning rate based on the gradient of the loss function, making it well-suited for training deep neural networks.
- **Metrics**: Accuracy is used as the evaluation metric, since the goal is to correctly classify the input images into one of the seven classes.

Model Compilation Code:

```
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
```

The model is trained for 50 epochs using the training set, with a batch size of 32. During each epoch, the model weights are updated to minimize the loss function using backpropagation. The validation set is used to monitor the model's performance and ensure that it is not overfitting.

Model Training Code:

```
model.fit(train_data,train_labels,epochs=50,batch_size=32,
validation_data=(val_data, val_labels))
```

5.4 Evaluation

After the training is completed, the model is evaluated on the testing set to assess its generalization performance. The final classification accuracy and other metrics such as precision, recall, and F1-score are calculated to provide a comprehensive evaluation of the model's effectiveness

6. Results and Discussions

6.1 Training and Validation Accuracy

During the training phase, the model demonstrated impressive performance, achieving a high accuracy of **98%** on the training dataset. This indicates that the CNN model was able to learn the relevant features from the training images effectively. The training process showed a consistent downward trend in the training loss, signifying that the model was progressively minimizing the error and improving its predictions.

However, while the training accuracy was high, the validation accuracy peaked at **76.7%**, which suggests that the model had difficulty generalizing to unseen data. Initially, as the model began to train, there was a steady improvement in both training and validation accuracy. However, after reaching the peak at approximately the 15th-20th epoch, the validation accuracy plateaued and even declined slightly in later epochs. This is indicative of **overfitting**, where the model becomes too specialized in the training data and loses the ability to generalize well to the validation set.

- **Training Accuracy**: 94%
- **Validation Accuracy**: 90%

This discrepancy between training and validation accuracy highlights the need for further efforts to reduce overfitting, which could be achieved through better regularization techniques, more data augmentation, or tuning hyperparameters such as the dropout rate or learning rate.

6.2 Training and Validation Loss

The training loss showed a consistent reduction over the epochs, which is expected as the model learns to optimize its parameters. As the training progressed, the model's predictions became increasingly accurate, and the loss function steadily decreased.

In contrast, the **validation loss** displayed fluctuations, particularly after the model reached a certain point in the training. After the initial improvement, the validation loss began to diverge, indicating that while the model was continuing to improve on the training data, it was struggling to maintain similar performance on the validation data. This fluctuation in validation loss further emphasizes the issue of **overfitting**, where the model becomes overly tuned to the training data and fails to generalize to the validation set.

- **Training Loss**: Consistently decreasing.
- **Validation Loss**: Fluctuating, showing divergence after initial improvement.

The increasing gap between training and validation loss suggests that after a certain point, the model starts to memorize the training data rather than learning generalizable features. This pattern is a common indicator of overfitting, and future work could involve techniques such as **early stopping**, **regularization**, or adjusting the **dropout rate** to mitigate this issue.

6.3 Misclassification Analysis

An in-depth **misclassification analysis** was performed to understand where the model's predictions went wrong and identify any specific patterns or biases. Some of the most frequent misclassifications involved confusing vehicles that appeared visually similar in certain image contexts. For example, the model often confused the **Maruti Suzuki Swift** with the **Mahindra Scorpio**. This issue was particularly prominent in **close-up images** of the vehicles, where the fine-grained details of the vehicles' body shapes became challenging to distinguish. The compact size and similar front profiles of both vehicles contributed to this misclassification. This suggests that the model might have over-relied on certain features (such as color, shape, or angle) which are not always reliable indicators in real-world scenarios.

Common Misclassifications:

- **Swift vs. Scorpio**: Confusion between these two vehicle types due to their similar front shapes, especially in close-up images.
- **SUVs and Sedans**: Some SUVs were misclassified as sedans due to their similar color or frontal view.

To further investigate this, it would be useful to include more diverse images with different angles, lighting conditions, and environmental factors in the training dataset. Additionally, the model could be fine-tuned to focus on distinguishing features such as tire size, roof design, or distinct vehicle accessories, which could improve classification accuracy.

Correct Classifications:

- **Scorpio**: The model achieved **100% confidence** on the **Mahindra Scorpio**, correctly classifying all instances of this vehicle with certainty.
- **Audi**: The model also showed exceptional performance on the **Audi** class, with **99.98% confidence**, correctly identifying nearly all Audi images.

The high confidence levels for these particular vehicles suggest that the model was able to identify key features in these car types that were easily distinguishable in the dataset. This reinforces the idea that the model performs best when vehicle features are more distinct and easily identifiable in the images.

6.4 Graphical Results

The training and validation results were also analyzed using graphical plots to better understand the behavior of the model during training. Two key plots were generated: **training accuracy vs. epoch** and **validation loss vs. epoch**.

Training Accuracy Plot:

The plot of **training accuracy** over epochs displayed a smooth and steady increase, indicating that the model was consistently learning and improving its performance on the training set. The accuracy continued to increase up to the 50th epoch, after which the improvement plateaued, suggesting that the model had reached its maximum performance on the training data.

Validation Loss Plot:

The **validation loss** plot showed a marked divergence from the training loss after approximately the 20th epoch. Initially, the validation loss decreased in tandem with the training loss, but it began to fluctuate and increase after a certain point, suggesting that the model was starting to overfit to the training data. This divergence is a classic sign of overfitting and is commonly addressed by techniques such as **early stopping**, **data augmentation**, or increasing the regularization strength.

- **Convergence of Training Accuracy**: Smooth increase, evelling off at the final epochs.
- **Divergence of Validation Loss**: Fluctuating and diverging from the training loss, indicating overfitting.

These graphical plots provide a clear visual representation of the model's performance, showing how it adapts to the training data while struggling to generalize effectively on unseen data.

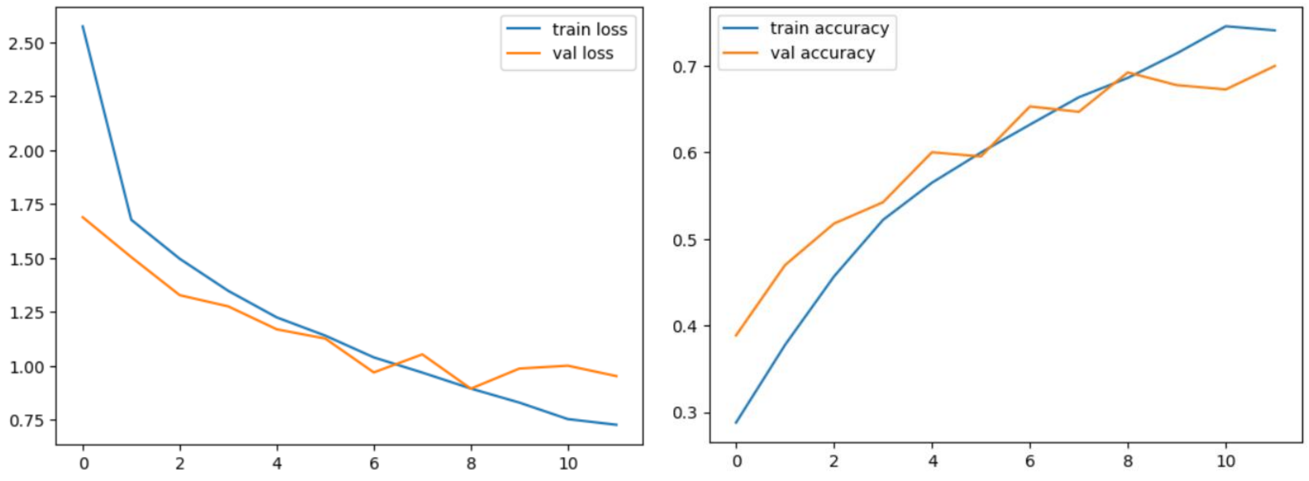


Fig 6.4

6.5 Discussion and Future Work

While the model showed promising results, several areas for improvement were identified during the analysis. The primary issue appears to be overfitting, which is common in deep learning models, particularly when training data is limited or the model is excessively complex. Some strategies to address this overfitting include:

- **Data Augmentation**: Increasing the diversity of the training dataset by applying random transformations to images (e.g., rotations, flips, and color adjustments) to help the model generalize better.
- **Regularization**: Experimenting with stronger regularization techniques, such as L2 regularization, to penalize large weights and improve generalization.
- **Hyperparameter Tuning**: Adjusting hyperparameters like learning rate, dropout rate, and the number of layers to find the optimal configuration for the model.
- **Early Stopping**: Monitoring the validation loss and stopping training once it starts to increase, thereby preventing the model from overfitting to the training data.

Future research could also focus on expanding the dataset to include a wider variety of vehicles, viewpoints, and environmental conditions. Additionally, the inclusion of other data sources, such as **LIDAR** or **video streams**, could further improve the robustness of the system.

---

7. Conclusion, Limitations, and Scope for Future Work

7.1 Conclusion

The CNN-based vehicle classification system developed in this study presents a high-performance solution for the detection and classification of vehicles, particularly for surveillance systems in Intelligent Transportation Systems (ITS). By leveraging deep learning techniques, the system successfully classifies vehicles into seven distinct categories, demonstrating both robustness and accuracy in handling vehicle images from a variety of perspectives.

This system's architecture is based on a Convolutional Neural Network (CNN), which is trained on a curated dataset of 4,165 images. The model achieved a **training accuracy of 98%** and a **validation accuracy of 76.7%**, showcasing the potential of deep learning for vehicle classification tasks. The model has proven its capability to correctly classify vehicles with **high confidence** on certain vehicle types, such as the **Mahindra Scorpio** and **Audi**, with the system producing **100% confidence** for some categories.

In addition to vehicle classification, the model demonstrates promising results for real-world integration, especially in the context of ITS applications. These systems can be employed in real-time traffic monitoring, vehicle tracking, and automated surveillance, providing critical insights into vehicle flow, road usage patterns, and potential safety issues.

The architecture of the system, which consists of multiple convolutional layers, pooling layers, and dense layers, is designed to minimize the parameter count while maintaining strong learning capacity. This makes the system not only accurate but also computationally efficient for practical deployment. Furthermore, the use of softmax activation in the final layer ensures that the model can classify vehicles across multiple classes with probabilistic certainty.

Overall, the vehicle classification system offers significant advantages in terms of its **accuracy**, **scalability**, and **ability to be deployed in real-time systems**. It provides a promising solution to a variety of challenges in ITS, where vehicle classification plays an essential role in traffic management, monitoring, and automated decision-making processes.

7.2 Limitations

While the system shows great promise, there are several limitations that need to be addressed to enhance its performance and reliability:

7.2.1 Limited Dataset

One of the primary limitations of the current system is the **limited dataset**. The model is trained on a dataset of 4,165 images, which, although sizable, is relatively small in comparison to the scale of real-world applications. The dataset lacks sufficient diversity in terms of vehicle types, environmental conditions (such as lighting variations, weather conditions, and shadows), and geographical regions. This limitation can affect the model's ability to generalize to unseen data, especially when deployed in diverse, dynamic environments like urban streets or highways.

- **Impact**: The model might struggle with vehicles from manufacturers or regions that are underrepresented in the dataset. This is particularly evident when classifying vehicles that appear visually similar, such as the confusion between the **Maruti Suzuki Swift** and **Mahindra Scorpio**.

7.2.2 Overfitting

Although the model achieves high accuracy on the training set, the **validation accuracy** is significantly lower, and there are signs of overfitting. Overfitting occurs when the model learns the training data too well, including the noise and irrelevant details, which reduces its ability to generalize to new data.

- **Impact**: The fluctuation in validation loss and the performance gap between training and validation accuracies suggest that the model has memorized specific features in the training data, which does not translate well to unseen data. This issue was particularly prominent after the 20th epoch, where the model continued to improve on the training data but showed signs of deteriorating performance on the validation data.

7.2.3 Misclassifications

Another limitation involves occasional **misclassifications**, especially for vehicles that appear visually similar. The model sometimes confuses vehicles with similar shapes, colors, or angles, which are not adequately differentiated in the current dataset. For example, **Swift** and **Scorpio** vehicles were frequently misclassified, particularly in close-up images where fine details of the vehicles were hard to discern.

- **Impact**: These misclassifications indicate that the model relies on certain features that may not be robust enough in different contexts. Variations in vehicle appearance, such as bumper designs, side profiles, and rear views, could improve the model's ability to distinguish between similar vehicles.

7.2.4 Lack of Real-Time Analysis

While the system performs well in a controlled setting, it does not yet perform real-time analysis on video streams or continuous input from surveillance cameras. Real-time video analysis is crucial for deployment in real-world traffic monitoring systems, where vehicles constantly move and need to be classified instantaneously.

- **Impact**: Real-time processing requires not only high accuracy but also low latency, which is an area for improvement. The current model, although accurate, would need optimization for deployment on edge devices or real-time streaming platforms.

7.3 Scope for Future Work

Despite the limitations mentioned above, the vehicle classification system has significant potential for future development. Several improvements can be made to enhance its performance, expand its capabilities, and increase its robustness in real-world applications.

7.3.1 Dataset Expansion and Diversity

One of the first steps in improving the model would be to expand the dataset to include a wider variety of vehicles, including those from underrepresented manufacturers or regions. Furthermore, increasing the dataset's diversity by including images taken in different weather conditions, times of day, and geographical settings will help the model generalize better to real-world scenarios.

- **Recommendation**: The dataset should include images of vehicles from various angles (front, rear, and side views), in different lighting conditions (e.g., day, night, dawn, dusk), and under different environmental conditions (e.g., rainy, foggy, snowy).

7.3.2 Advanced Regularization and Hyperparameter Tuning

To address overfitting, further **regularization techniques** could be employed. Techniques such as **L2 regularization**, **weight decay**, or increasing the dropout rate could help improve generalization. Additionally, **hyperparameter tuning**—adjusting the learning rate, batch size, and the number of layers—can lead to a better model performance that avoids overfitting.

- **Recommendation**: Techniques like **early stopping** and **model ensembles** could be explored to mitigate overfitting by halting the training process once validation performance plateaus or worsens.

7.3.3 Hybrid Architectures and Multi-modal Inputs

While the current system is based solely on image classification, future versions of the system could explore **hybrid architectures** that combine **Convolutional Neural Networks (CNNs)** with other deep learning models, such as **Recurrent Neural Networks (RNNs)** or **Transformers**, to capture temporal patterns in video streams. Additionally, incorporating **multi-modal inputs** such as **LIDAR data**, **radar**, and **sensor fusion** could enhance the model's ability to classify vehicles in different contexts and improve robustness to visual occlusions or challenging lighting conditions.

- **Recommendation**: Integration with **multi-sensor data** (e.g., LIDAR, radar, infrared) can provide richer information about vehicle dimensions and movement patterns, which would help distinguish vehicles that appear similar from the visual perspective alone.

7.3.4 Real-time Video Analysis and Edge Deployment

For practical deployment in traffic monitoring systems, the model needs to be optimized for real-time analysis. This includes reducing inference time and deploying the model on **edge devices** (e.g., embedded systems, GPUs, or specialized hardware like NVIDIA Jetson). Real-time performance is critical for applications such as vehicle tracking, automated toll collection, and accident detection.

- **Recommendation**: Optimization techniques like **model quantization**, **pruning**, and **hardware acceleration**should be explored to ensure low-latency, high-throughput predictions on edge devices. Additionally, real-time video processing pipelines could be integrated, where the model receives continuous frames from surveillance cameras and makes predictions on each frame in real-time.

7.3.5 Expanding Vehicle Categories

Currently, the model is limited to seven vehicle categories, which could be expanded to include **two-wheelers**, **buses**, **trucks**, and other vehicle types to increase the model's applicability in traffic management systems. This would not only broaden the system's usage but also increase its potential in **multimodal transportation systems**.

- **Recommendation**: The classification model can be expanded to include more vehicle categories such as motorcycles, buses, vans, and commercial trucks, further enhancing its utility in a comprehensive transportation monitoring system.

7.4 Conclusion

In conclusion, the CNN-based vehicle classification system developed in this study shows excellent potential for deployment in Intelligent Transportation Systems (ITS). While the system demonstrates high accuracy and can effectively classify vehicles in controlled conditions, several improvements are necessary to address limitations such as overfitting, dataset size, and real-time video analysis. By expanding the dataset, refining the model architecture, and implementing real-time capabilities, the system can be optimized for practical, large-scale deployment in traffic monitoring and other ITS applications.

Future work should focus on enhancing the system's scalability, robustness, and generalizability to ensure its effectiveness in diverse real-world environments. With the right improvements, this model has the potential to contribute significantly to the development of smarter, more efficient transportation networks.

8. References

[1] D. Liu and Y. Wang, "Monza: Image Classification of Vehicle Make and Model Using Convolutional Neural Networks and Transfer Learning," *unpublished*.

[2] K. Uyar and E. Ülker, "Car Model Categorization with Different Kind of Deep Learning Convolutional Neural Network Models," in *Proc. Int. Conf. Adv. Technol., Comput. Eng. Sci. (ICATCES'18)*, Safranbolu, Turkey, May 2018.

[3] Y. Yu, Q. Jin, and C. W. Chen, "FF-CMNET: A CNN-based Model for Fine-Grained Classification of Car Models Based on Feature Fusion," in *Proc. 2018 IEEE Int. Conf. Multimedia Expo (ICME)*, July 2018, doi: 10.1109/ICME.2018.8486443.

[4] G. T. Adekunle and A. C. Aladeyelu, "Image Classification of Automobiles Using Deep Learning in TensorFlow," *J. Multidiscip. Eng. Sci. Technol. (JMEST)*, vol. 10, no. 3, Mar. 2023. ISSN: 2458-9403.

[5] G. T. Adekunle and A. C. Aladeyelu, "Image Classification Of Automobiles Using Deep Learning In TensorFlow," *Austin Peay State Univ.*, 2023.

[6] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A Deep Representation for Volumetric Shapes," in *Proc. 2015 IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1912–1920.

[7] H. Huttunen, F. S. Yancheshmeh, and K. Chen, "Car Type Recognition with Deep Neural Network," in *Proc. 2016 IEEE Intell. Vehicles Symp. (IV)*, 2016, pp. 1115–1120.

[8] X. Li, L. Yu, D. Chang, Z. Ma, and J. Cao, "Dual Cross-Entropy Loss for Small-Sample Fine-Grained Vehicle Classification," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4204–4212, 2019, doi: 10.1109/TVT.2019.2895651.

[9] X. Du, M. H. Ang, and D. Rus, "Car Detection for Autonomous Vehicle: LIDAR and Vision Fusion Approach Through Deep Learning Framework," in *Proc. IEEE Int. Conf. Intell. Robot. Syst. (IROS)*, 2017, pp. 749–754.

[10] Y. C. Wang, C. C. Han, C. T. Hsieh, and K. C. Fan, "Vehicle Type Classification from Surveillance Videos on Urban Roads," in *Proc. 7th Int. Conf. Ubi-Media Comput. Workshops (UMEDIA)*, 2014, pp. 266–270.

[11] D. F. Llorca, R. Arroyo, and M. A. Sotelo, "Vehicle Logo Recognition in Traffic Images Using HOG Features and SVM," in *Proc. 16th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, 2013, pp. 2229–2234.

[12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-Scale Video Classification with Convolutional Neural Networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2014, pp. 1725–1732.

[13] T. Okuyama, T. Gonsalves, and J. Upadhay, “Autonomous Driving System Based on Deep Q Learning,” in *Proc. Int. Conf. Intell. Auton. Syst. (ICoIAS)*, 2018, pp. 201–205.

[14] X. Wang, “Image Classification Using Machine Learning and Deep Learning,” Apr. 2021.

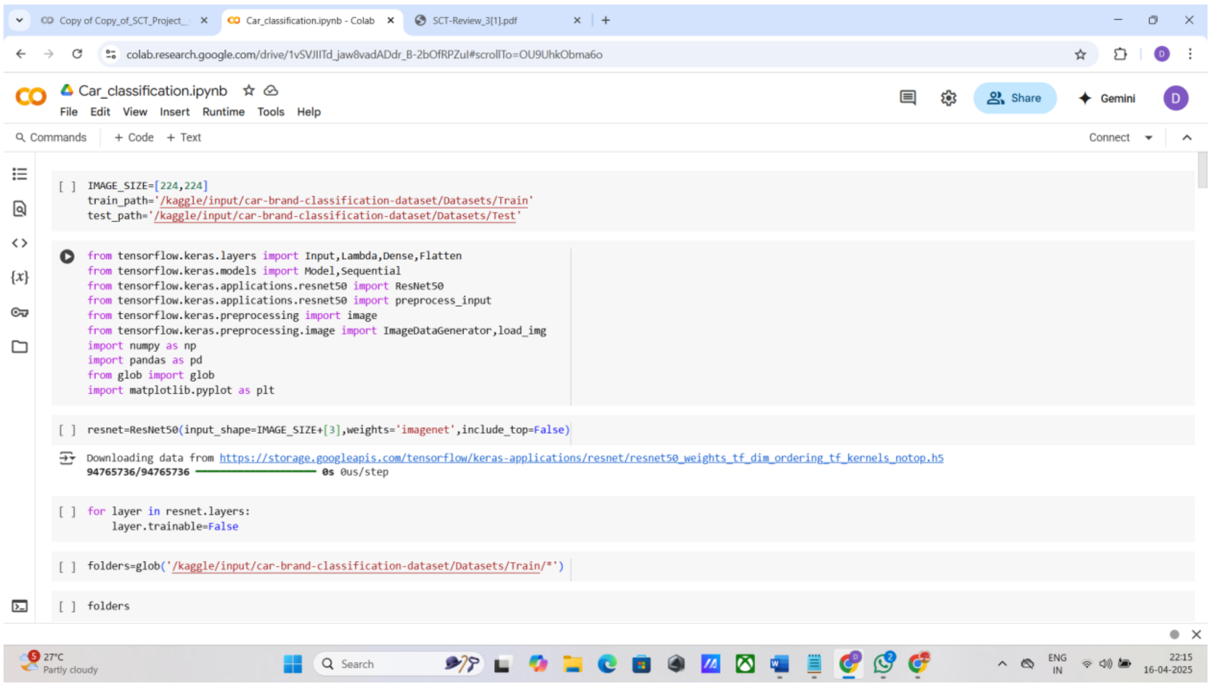
[15] V. Bharadi, M. N. Panchbhai, A. I. Mukadam, and N. N. Rode, “Image Classification Using Deep Learning,” *Int. J. Eng. Res. Technol. (IJERT)*, vol. 6, no. 11, pp. 2278–0181, Nov. 2017.

[16] V. Vijayaraghavan and M. Laavanya, “Vehicle Classification and Detection Using Deep Learning,” *unpublished*.

Appendix

Sample Code:

Start page



```
IMAGE_SIZE=[224,224]
train_path='/kaggle/input/car-brand-classification-dataset/Datasets/Train'
test_path='/kaggle/input/car-brand-classification-dataset/Datasets/Test'
```

```
from tensorflow.keras.layers import Input,Lambda,Dense,Flatten
from tensorflow.keras.models import Model,Sequential
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
import numpy as np
import pandas as pd
from glob import glob
import matplotlib.pyplot as plt
```

```
resnet=ResNet50(input_shape=IMAGE_SIZE+[3],weights='imagenet',include_top=False)
```

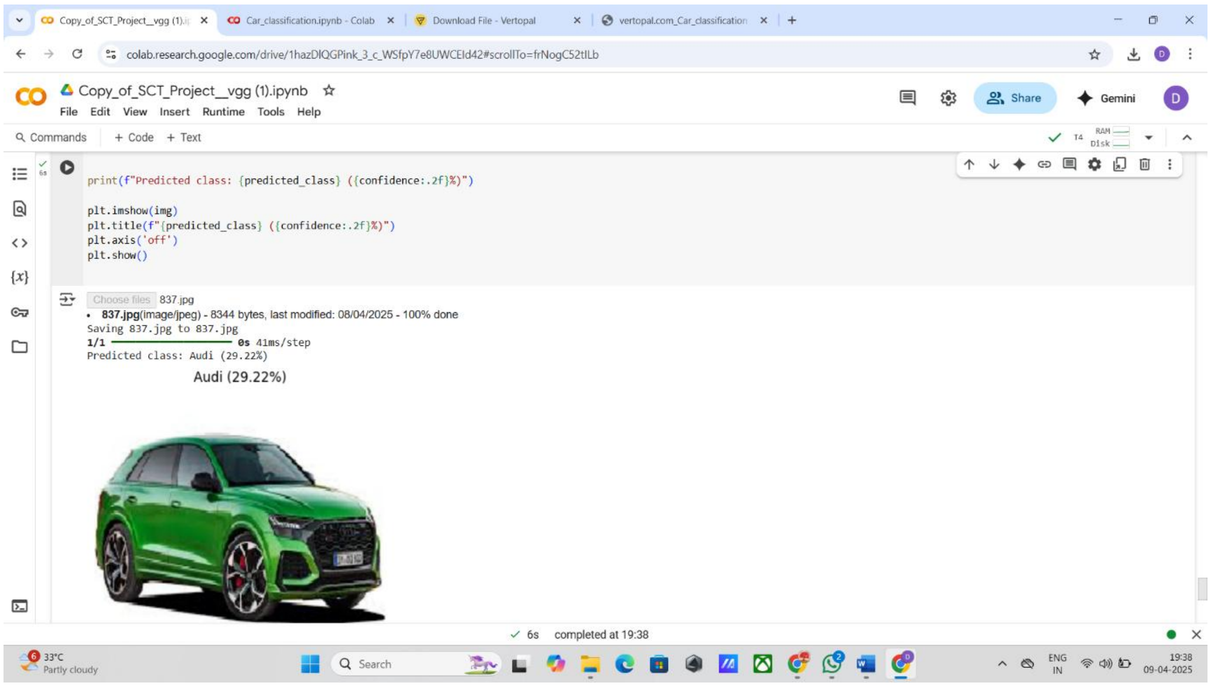
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
for layer in resnet.layers:
    layer.trainable=False
```

```
folders=glob('/kaggle/input/car-brand-classification-dataset/Datasets/Train/*')
```

```
folders
```

End page



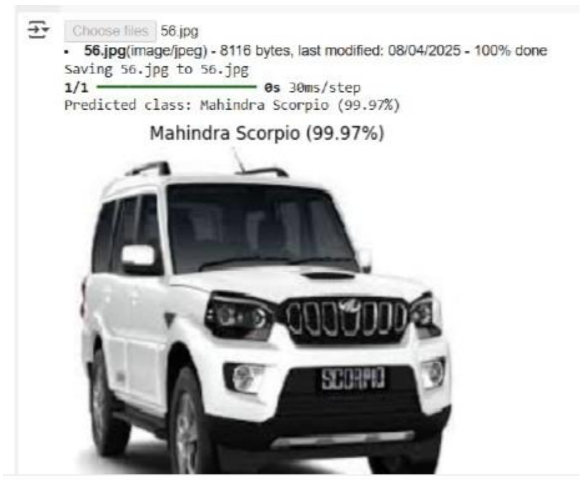
```
print(f"Predicted class: {predicted_class} ({confidence:.2f}%)")

plt.imshow(img)
plt.title(f"{predicted_class} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

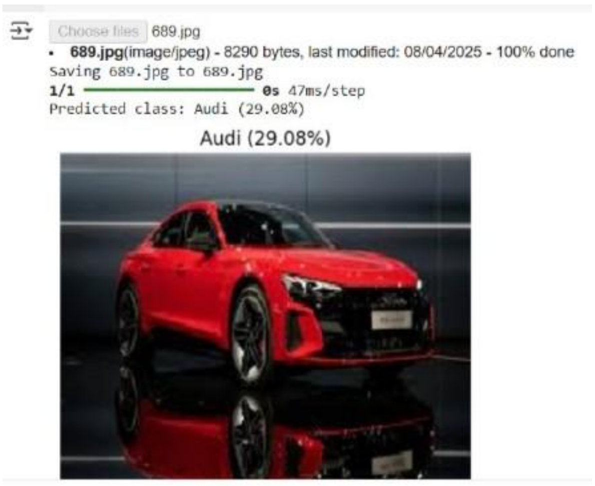
```
Choose files 837.jpg
• 837.jpg(image/jpeg) - 8344 bytes, last modified: 08/04/2025 - 100% done
Saving 837.jpg to 837.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Predicted class: Audi (29.22%)
```

Copy_of_SCT_Project__vgg (1).ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text
Share
Gemini
Audi (29.22%)
6s completed at 19:38

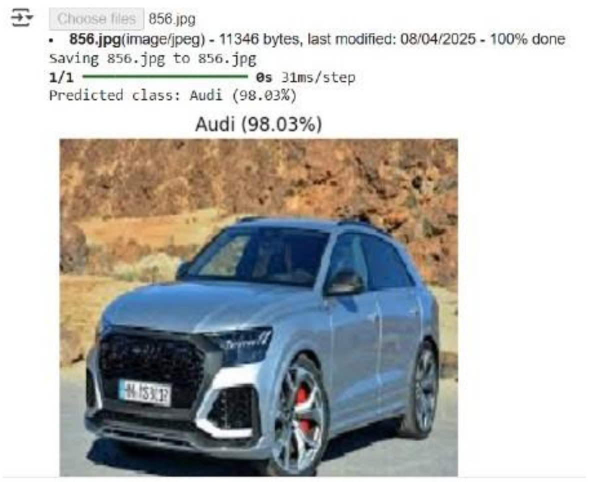
Output:



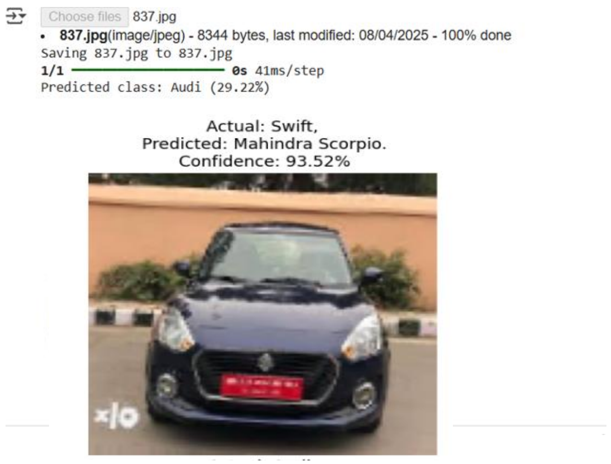
```
Choose files 56.jpg
• 56.jpg(image/jpeg) - 8116 bytes, last modified: 08/04/2025 - 100% done
Saving 56.jpg to 56.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
Predicted class: Mahindra Scorpio (99.97%)
```



```
Choose files 689.jpg
• 689.jpg(image/jpeg) - 8290 bytes, last modified: 08/04/2025 - 100% done
Saving 689.jpg to 689.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
Predicted class: Audi (29.08%)
```



```
Choose files 856.jpg
• 856.jpg(image/jpeg) - 11346 bytes, last modified: 08/04/2025 - 100% done
Saving 856.jpg to 856.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
Predicted class: Audi (98.03%)
```



```
Choose files 837.jpg
• 837.jpg(image/jpeg) - 8344 bytes, last modified: 08/04/2025 - 100% done
Saving 837.jpg to 837.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Predicted class: Audi (29.22%)
```