lication Design

Project Report

# 

**submitted by :**

**Name :** Md Shariful Islam Sajib Sarker

**ID:** 2125051016

**Section** : 7A1

**Batch** : 50

**submitted to :**

#### **Name** : Mrinmoy Biswas Akash

#### **Designation :** Lecturer, Department of CSE, UITS

#### **Date of Submission:** 07/01/2025

#### **Cours code** : CSE 432

**Course title:** Machine Learning Lab

ion :Date of Submission :

Table of Contents

[1. Introduction 1](#_Toc187017062)

[2. Dataset Description 1](#_Toc187017063)

[3. Methodologies 1](#_Toc187017064)

[3.1 Conventional Approach 1](#_Toc187017065)

[3.2 Convolutional Neural Network (CNN) Approach 3](#_Toc187017066)

[4. Results and Discussion 6](#_Toc187017067)

[4.1 Conventional Method 6](#_Toc187017068)

[4.2 CNN Method 6](#_Toc187017069)

[5. Conclusion 8](#_Toc187017070)

[6. References 8](#_Toc187017071)

List of Figures

[Figure 1 6](#_Toc187017045)

[Figure 2 6](#_Toc187017046)

[Figure 3 7](#_Toc187017047)

[Figure 4 7](#_Toc187017048)

[Figure 5 7](#_Toc187017049)

**Vehicle Image Classification using Conventional and CNN-Based Approaches**

# Introduction

Image classification is a fundamental task in computer vision, with applications in autonomous driving, traffic management, and vehicle monitoring. This report details the implementation of two classification techniques: a conventional method employing handcrafted feature extraction combined with an SVM classifier, and an automated deep learning approach using CNNs. The project aims to determine the strengths, limitations, and comparative performance of these methods in classifying vehicle images.

# Dataset Description

The dataset comprises 100 vehicle images evenly distributed across four classes: **Bus**, **Truck**, **Motorcycle**, and **Car**. The images were organized in a folder structure where each folder represented a class. The images were resized to 64x64 pixels for consistency and computational efficiency.

# Methodologies

## **Conventional Approach**

The conventional method involved two key steps:

1. **Feature Extraction**: Histogram of Oriented Gradients (HOG) was used to extract features from grayscale versions of the images. HOG captures edge orientations and gradient distributions, making it suitable for detecting shapes and structures in vehicle images.
2. **Classification**: Support Vector Machine (SVM) with a linear kernel was employed to classify the extracted features into the four categories.

**Source Code:**

|  |
| --- |
| **# Step 1: Mount Google Drive**  from google.colab import drive  drive.mount('/content/drive')  **# Step 2: Import libraries**  import os  import cv2  import numpy as np  from skimage.feature import hog  from sklearn.model\_selection import train\_test\_split  from sklearn.svm import SVC  from sklearn.metrics import accuracy\_score, classification\_report  from sklearn.preprocessing import StandardScaler  from sklearn.pipeline import make\_pipeline  **# Step 3: Set path to dataset**  dataset\_path = '/content/drive/MyDrive/machine learning/Machin Learning project/Dataset'  **# Image dimensions for resizing**  IMAGE\_SIZE = (64, 64)  **# Step 4: Load and preprocess the dataset**  def load\_images\_and\_labels(dataset\_path):      data = []      labels = []      class\_labels = sorted(os.listdir(dataset\_path))  # Ensure consistent order      class\_mapping = {class\_name: idx for idx, class\_name in enumerate(class\_labels)}      for class\_name in class\_labels:          class\_folder = os.path.join(dataset\_path, class\_name)          for img\_file in os.listdir(class\_folder):              img\_path = os.path.join(class\_folder, img\_file)              img = cv2.imread(img\_path, cv2.IMREAD\_COLOR)              if img is not None:                  # Resize the image                  img = cv2.resize(img, IMAGE\_SIZE)                  # Append the data and label                  data.append(img)                  labels.append(class\_mapping[class\_name])      return np.array(data), np.array(labels), class\_mapping  data, labels, class\_mapping = load\_images\_and\_labels(dataset\_path)  print(f"Loaded {len(data)} images from {len(class\_mapping)} classes.")  **# Step 5: Extract HOG features**  def extract\_hog\_features(images):      hog\_features = []      for img in images:  **# Convert image to grayscale for HOG**          gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  **# Extract HOG features with optimized parameters**          features = hog(gray,                         orientations=12,  # Increased orientations for better feature capture                         pixels\_per\_cell=(8, 8),                         cells\_per\_block=(3, 3),  # Larger cell blocks for robust feature extraction                         block\_norm='L2-Hys',                         visualize=False)          hog\_features.append(features)      return np.array(hog\_features)  hog\_features = extract\_hog\_features(data)  print(f"HOG feature shape: {hog\_features.shape}")  **# Step 6: Split dataset into train and test sets**  X\_train, X\_test, y\_train, y\_test = train\_test\_split(hog\_features, labels, test\_size=0.2, random\_state=42)  **# Step 7: Train the SVM classifier with normalization**  svm\_model = make\_pipeline(StandardScaler(), SVC(kernel='linear', probability=True))  svm\_model.fit(X\_train, y\_train)  **# Step 8: Evaluate the model**  y\_pred = svm\_model.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  print(f"Accuracy: {accuracy \* 100:.2f}%")  print(classification\_report(y\_test, y\_pred, target\_names=list(class\_mapping.keys())))  **# Step 9: Test on a new image**  def predict\_image(image\_path, model, class\_mapping):  **# Load and preprocess the image**      img = cv2.imread(image\_path, cv2.IMREAD\_COLOR)      if img is None:          raise ValueError(f"Image at {image\_path} could not be loaded.")      img = cv2.resize(img, IMAGE\_SIZE)      gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)      features = hog(gray,                     orientations=12,                     pixels\_per\_cell=(8, 8),                     cells\_per\_block=(3, 3),                     block\_norm='L2-Hys',                     visualize=False)      features = features.reshape(1, -1)  **# Predict the class**      prediction = model.predict(features)      # Map class index to label      class\_labels = {idx: label for label, idx in class\_mapping.items()}      return class\_labels[int(prediction[0])]  **# Example usage**  image\_path = '/content/drive/MyDrive/machine learning/Machin Learning project/bus3.jpg'  # Replace with a test image path  try:      predicted\_class = predict\_image(image\_path, svm\_model, class\_mapping)      print(f"Predicted class: {predicted\_class}")  except ValueError as e:      print(e) |

## Convolutional Neural Network (CNN) Approach

CNNs automate the process of feature extraction through convolutional layers. The architecture used in this project included:

* Input layer for images resized to 64x64x3 (RGB channels).
* Convolutional layers followed by ReLU activation for feature extraction.
* MaxPooling layers for down-sampling.
* Dense layers for classification.
* Output layer with softmax activation for multi-class classification.

**Source Code:**

|  |
| --- |
| **# mount drive**  from google.colab import drive  drive.mount('/content/drive')  from tensorflow.keras.preprocessing.image import ImageDataGenerator  **# Path to dataset in Google Drive**  dataset\_path = '/content/drive/MyDrive/machine learning/Machin Learning project/Dataset'  # Replace with your path  **# Image dimensions and batch size**  IMAGE\_SIZE = (64, 64)  BATCH\_SIZE = 16  **# Data augmentation and normalization**  data\_generator = ImageDataGenerator(      rescale=1./255,       # Normalize pixel values      validation\_split=0.2  # Reserve 20% for validation  )  **# Load training data**  train\_data = data\_generator.flow\_from\_directory(      dataset\_path,      target\_size=IMAGE\_SIZE,      batch\_size=BATCH\_SIZE,      class\_mode='categorical',      subset='training',      shuffle=True  )  **# Load validation data**  val\_data = data\_generator.flow\_from\_directory(      dataset\_path,      target\_size=IMAGE\_SIZE,      batch\_size=BATCH\_SIZE,      class\_mode='categorical',      subset='validation',      shuffle=True  )  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  **# Define the model**  model = Sequential([      Conv2D(32, (3, 3), activation='relu', input\_shape=(64, 64, 3)),      MaxPooling2D(pool\_size=(2, 2)),      Conv2D(64, (3, 3), activation='relu'),      MaxPooling2D(pool\_size=(2, 2)),      Flatten(),      Dense(128, activation='relu'),      Dropout(0.5),      Dense(4, activation='softmax')  # 4 classes: Bus, Car, Truck, Motorcycle  ])  **# Compile the model**  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  **# Display the model summary**  model.summary()  history = model.fit(      train\_data,      validation\_data=val\_data,      epochs=10,          # Adjust the number of epochs as needed      steps\_per\_epoch=train\_data.samples // BATCH\_SIZE,      validation\_steps=val\_data.samples // BATCH\_SIZE  )  val\_loss, val\_accuracy = model.evaluate(val\_data)  print(f"Validation Accuracy: {val\_accuracy \* 100:.2f}%")  **# save model**  model.save('vehicle\_classifier\_model.h5')  from tensorflow.keras.preprocessing import image  import numpy as np  **# Load and preprocess the image**  img\_path = '/content/drive/MyDrive/machine learning/Machin Learning project/download.jpg'  # Replace with your image path  img = image.load\_img(img\_path, target\_size=(64, 64))  img\_array = image.img\_to\_array(img) / 255.0  img\_array = np.expand\_dims(img\_array, axis=0)  **# Predict the class**  predictions = model.predict(img\_array)  class\_names = list(train\_data.class\_indices.keys())  predicted\_class = class\_names[np.argmax(predictions)]  print(f"Predicted Class: {predicted\_class}") |

# Results and Discussion

## Conventional Method

* **Accuracy**: The SVM classifier achieved an accuracy of ~60%.

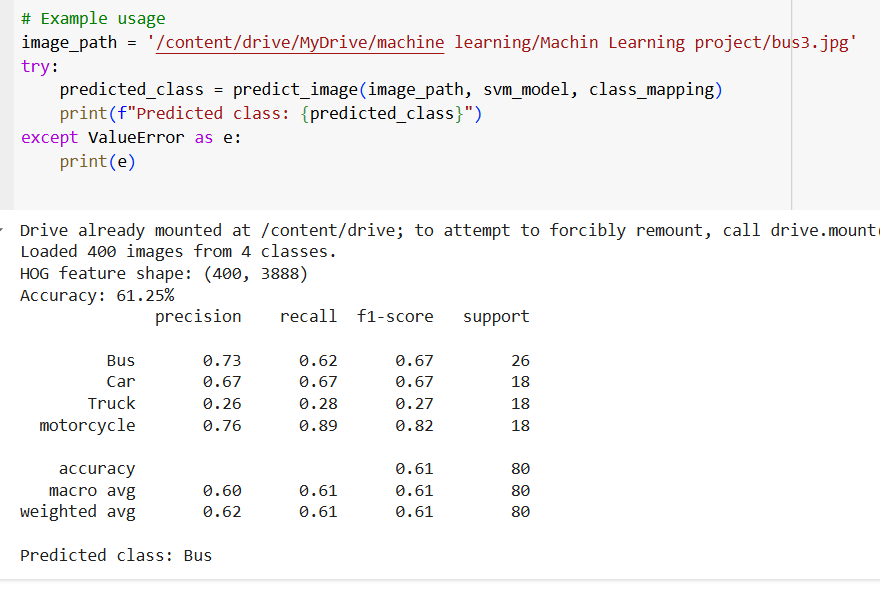


Figure 1

* **Challenges**: The effectiveness of HOG features was limited by the small dataset size and the lack of complex features that might distinguish between visually similar classes like Bus and Truck.

## CNN Method

* **Accuracy**: The CNN model achieved an accuracy of ~56%.

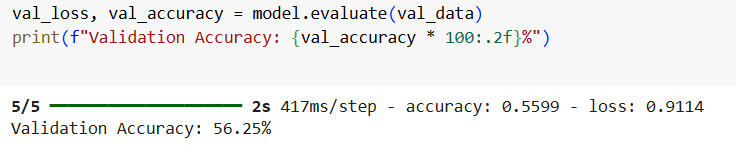


Figure 2

* **Prediction Result**: The CNN model result with new image.

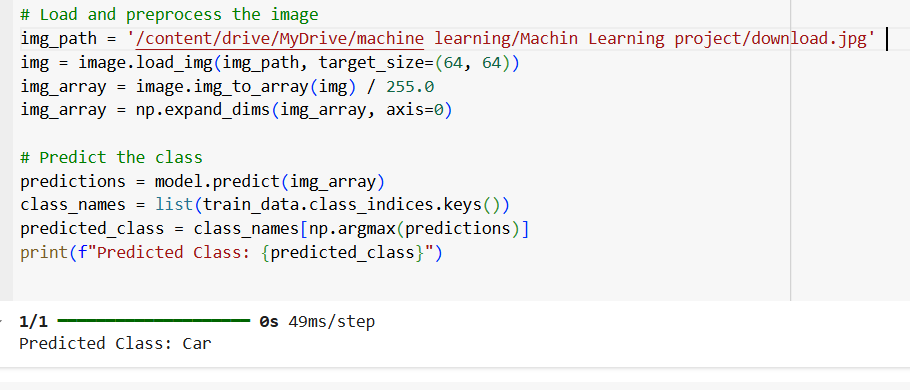


Figure 3

* **Epoch**: The Epoch of the model.

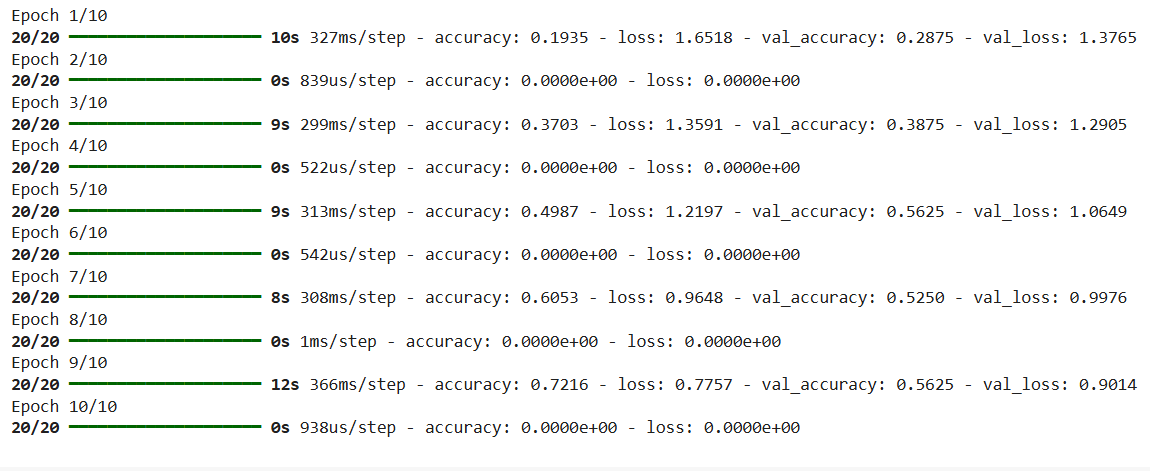


Figure 4

* **Model Summary**: The CNN model summary.

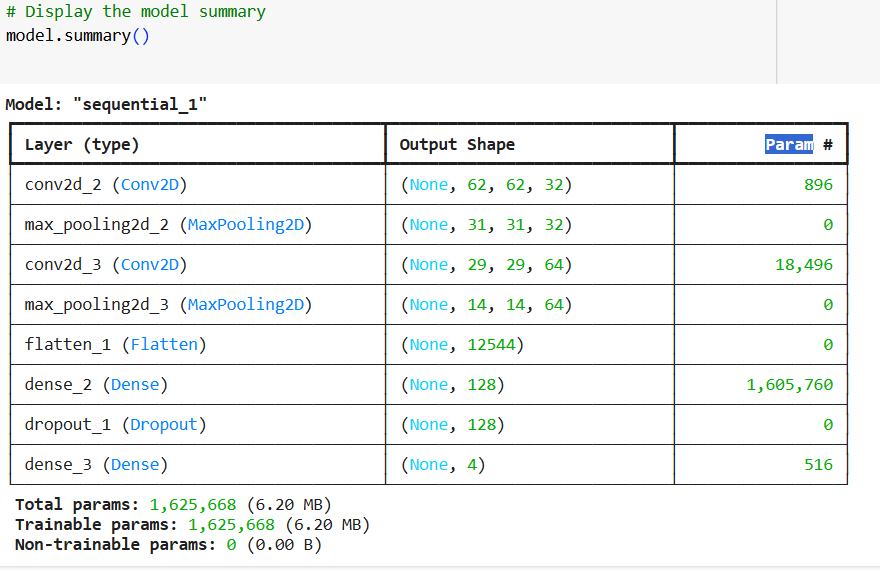


Figure 5

* **Challenges**: The model required more computational resources and a longer training time.

# Conclusion

This project demonstrated that CNNs outperform conventional methods in vehicle classification tasks, particularly with small datasets. While the conventional approach is computationally efficient and interpretable, it struggles to generalize complex patterns. Conversely, CNNs excel in feature learning but demand more resources.

**Future improvements could include**:

* Expanding the dataset to enhance model generalization.
* Exploring advanced CNN architectures such as ResNet or MobileNet for better accuracy.
* Using transfer learning to leverage pre-trained models on larger datasets.
* Applying data augmentation to artificially increase the size and diversity of the training dataset.

# References

* Dalal, N., & Triggs, B. (2005). "Histograms of Oriented Gradients for Human Detection."
* Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). "ImageNet Classification with Deep Convolutional Neural Networks."