**CHAPTER 1**

**INTRODUCTION**

Car model prediction, also known as vehicle recognition or car make and model identification, is a computer vision and machine learning task that involves automatically determining the make and model of a vehicle from an image or video feed. This technology has gained significance due to its diverse range of applications, including automotive industry needs, security and surveillance, traffic management, insurance and finance, and environmental monitoring. The history of car model prediction can be traced back to the following key developments such as early computer vision research that included efforts to recognize simple objects and shapes. While not specific to car models, these foundational studies laid the groundwork for more complex pattern recognition tasks. Next, emergence of machine learning in 1990s saw advancements in machine learning algorithms, particularly in the fields of neural networks and support vector machines (SVMs). These developments enabled researchers to explore more sophisticated image recognition techniques. Then, the availability of large-scale digital image datasets, including those containing images of vehicles, played a crucial role in advancing car model prediction. Researchers could train and test machine learning models on substantial and diverse datasets.

The use of deep CNNs for image classification and recognition has gained prominence in recent years, primarily due to breakthroughs in deep learning research. Historically, deep CNNs have their roots in neural network research dating back to the 1980s and 1990s. Deep Convolutional Neural Networks (CNNs) have emerged as a powerful tool in computer vision, particularly in image recognition and classification tasks. CNNs are designed to automatically learn features from raw data, making them highly effective for tasks involving image processing and pattern recognition. In the context of car model prediction using vehicle pattern recognition, deep CNNs can be employed to recognize and classify vehicles based on their visual features, such as shape, color, and texture. The need for a deep CNN model for car model prediction using vehicle pattern recognition arises from several factors such as large-scale data, automation since the manual car model identification and recognition are time-consuming and error prone. Hence, automation through deep learning models can significantly improve efficiency. Next, the accuracy, where the deep CNN models can achieve high levels of accuracy in recognizing car models, surpassing human capabilities in some cases. Finally, the versatility, where these models can handle a wide variety of vehicle types, brands, and conditions, making them versatile in different applications. In addition, car model prediction became a practical technology with widespread commercial applications. It found use in the automotive industry for inventory management, in security and surveillance for vehicle tracking, and in traffic management for monitoring and optimization. On the other hand, in today’s world, car model prediction is applied in various contexts, such as automatic toll collection systems, parking management, vehicle inventory management in car dealerships, vehicle recognition in smart cities, and more. Therefore, this project implements the deep CNN model for the prediction of a car model through the vehicle pattern recognition. It also provides the performance evaluation of existing machine learning models in terms of prediction accuracy.

**Significance**

Car model prediction using vehicle pattern recognition has significant implications in various domains:

* Automotive Industry: Deep CNN models can assist in automating the identification and categorization of vehicles, helping manufacturers, dealerships, and service centers manage their inventory and offer tailored services.
* Security and Surveillance: Law enforcement agencies and security companies can use these models for vehicle identification in surveillance videos, border control, and tracking suspicious vehicles.
* Traffic Management: Smart cities and transportation authorities can utilize these models to monitor and manage traffic by tracking vehicle types and patterns.
* Insurance and Finance: Car insurance companies can leverage vehicle recognition for risk assessment and fraud detection, while financial institutions may use it for assessing vehicle values.
* Environmental Monitoring: Tracking vehicle types and emissions can aid in assessing and mitigating the environmental impact of transportation.

**Problem Definition**

The problem addressed by the deep CNN model for car model prediction using vehicle pattern recognition can be defined as follows:

* Given: A dataset containing labeled images of vehicles representing different car models.
* Objective: To develop a deep CNN model that can accurately recognize and classify vehicles based on their visual features, ultimately predicting the specific car model present in a given image.

**Challenges**

* Handling variations in lighting, angle, and image quality.
* Dealing with occlusions and partial views of vehicles.
* Managing a diverse range of car models and brands.
* Training a model that generalizes well to unseen data.
* Balancing accuracy with computational efficiency, especially in real-time applications.

**SOURCE CODE**

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

import matplotlib.pyplot as plt

import numpy as np

from tkinter import ttk

from tkinter import filedialog

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from keras.utils.np\_utils import to\_categorical

from keras.models import Sequential

from keras.layers.core import Dense,Activation,Dropout, Flatten

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm

from sklearn.metrics import accuracy\_score

import cv2

from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

from sklearn.linear\_model import LogisticRegression

from PIL import ImageTk,Image

from sklearn.decomposition import PCA

main = Tk()

main.title("Machine Learning-Based Car Model Prediction through Vehicle Pattern Recognition")

w=main.winfo\_screenwidth()

h=main.winfo\_screenheight()

#imgTemp=Image.open("C:\\Users\\nehar\\Downloads\\truck.png")

imgTemp=Image.open("car2.webp")

img2=imgTemp.resize((w,h))

img=ImageTk.PhotoImage(img2)

label = Label(main,image=img)

label.place(height=h,width=w)

main.geometry("1300x1200")

global filename

global X, Y

global model

global X\_train, X\_test, y\_train, y\_test

accuracy = []

global XX

global classifier

names = ['AM General Hummer SUV 2000', 'Acura RL Sedan 2012', 'Acura TL Sedan 2012', 'Acura TL Type-S 2008', 'Acura TSX Sedan 2012']

def uploadDataset():

global filename

text.delete('1.0', END)

filename = filedialog.askdirectory(initialdir=".")

text.insert(END,'dataset loaded\n')

X = np.load("model/X.txt.npy")

Y = np.load("model/Y.txt.npy")

X = np.asarray(X)

Y = np.asarray(Y)

img = X[20].reshape(64,64,3)

cv2.imshow('ff',cv2.resize(img,(250,250)))

cv2.waitKey(0)

def linearKNN():

accuracy.clear()

text.delete('1.0', END)

X = np.load("model/X.txt.npy")

Y = np.load("model/Y.txt.npy")

print(X.shape)

print(Y.shape)

temp = X

XX = np.reshape(temp, (temp.shape[0],(temp.shape[1]\*temp.shape[2]\*temp.shape[3])))

pca = PCA(n\_components = 180)

XX = pca.fit\_transform(XX)

print(XX.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(XX, Y, test\_size=0.2)

cls = LogisticRegression()

cls.fit(X\_train, y\_train)

predict = cls.predict(X\_test)

#predict = np.argmax(predict, axis=1)

#y\_test1 = np.argmax(y\_test, axis=1)

acc = accuracy\_score(y\_test,predict)\*100

accuracy.append(acc)

text.insert(END,'Linear Regression Prediction Accuracy : '+str(acc)+"\n")

cls = KNeighborsClassifier()

cls.fit(X\_train, y\_train)

predict = cls.predict(X\_test)

acc = accuracy\_score(y\_test,predict)\*100

accuracy.append(acc)

text.insert(END,'KNN Prediction Accuracy : '+str(acc)+"\n")

bars = ('Linear Regression', 'KNN')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, [accuracy[0],accuracy[1]])

plt.xticks(y\_pos, bars)

plt.show()

plt.title('Linear Regression & KNN Accuracy Performance Graph')

plt.show()

def SVMCNN():

global classifier

X = np.load("model/X.txt.npy")

Y = np.load("model/Y.txt.npy")

print(X.shape)

print(Y.shape)

temp = X

XX = np.reshape(temp, (temp.shape[0],(temp.shape[1]\*temp.shape[2]\*temp.shape[3])))

pca = PCA(n\_components = 180)

XX = pca.fit\_transform(XX)

print(XX.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(XX, Y, test\_size=0.2)

cls = svm.SVC()

cls.fit(X\_train, y\_train)

predict = cls.predict(X\_test)

acc = accuracy\_score(y\_test,predict)\*100

accuracy.append(acc)

text.insert(END,'SVM Prediction Accuracy : '+str(acc)+"\n")

Y1 = to\_categorical(Y)

cnn = Sequential()

cnn.add(Convolution2D(32, 3, 3, input\_shape = (64, 64, 3), activation = 'relu'))

cnn.add(MaxPooling2D(pool\_size = (2, 2)))

cnn.add(Convolution2D(32, 3, 3, activation = 'relu'))

cnn.add(MaxPooling2D(pool\_size = (2, 2)))

cnn.add(Flatten())

cnn.add(Dense(output\_dim = 256, activation = 'relu'))

cnn.add(Dense(output\_dim = 5, activation = 'softmax'))

print(cnn.summary())

cnn.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

hist = cnn.fit(X, Y1, batch\_size=16, epochs=10, shuffle=True, verbose=2)

cnn\_history = hist.history

cnn\_history = cnn\_history['accuracy']

acc = cnn\_history[9] \* 100

accuracy.append(acc)

text.insert(END,'CNN Prediction Accuracy : '+str(acc)+"\n\n")

classifier = cnn

bars = ('SVM','CNN')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, [accuracy[2],accuracy[3]])

plt.xticks(y\_pos, bars)

plt.show()

plt.title('SVM & CNN Accuracy Performance Graph')

plt.show()

def KNNSVM():

X = np.load("model/X.txt.npy")

Y = np.load("model/Y.txt.npy")

print(X.shape)

print(Y.shape)

temp = X

XX = np.reshape(temp, (temp.shape[0],(temp.shape[1]\*temp.shape[2]\*temp.shape[3])))

pca = PCA(n\_components = 180)

XX = pca.fit\_transform(XX)

print(XX.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(XX, Y, test\_size=0.2)

cls = KNeighborsClassifier()

cls.fit(X\_train, y\_train)

predict = cls.predict(X\_test)

acc = accuracy\_score(y\_test,predict)\*100

accuracy.append(acc)

text.insert(END,'KNN Prediction Accuracy : '+str(acc)+"\n")

cls = svm.SVC()

cls.fit(X\_train, y\_train)

predict = cls.predict(X\_test)

acc = accuracy\_score(y\_test,predict)\*100

accuracy.append(acc)

text.insert(END,'SVM Prediction Accuracy : '+str(acc)+"\n")

bars = ('KNN Inference','SVM Inference')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, [accuracy[4],accuracy[5]])

plt.xticks(y\_pos, bars)

plt.show()

plt.title('KNN & SVM Inference Accuracy Performance Graph')

plt.show()

def KNNCNN():

X = np.load("model/X.txt.npy")

Y = np.load("model/Y.txt.npy")

print(X.shape)

print(Y.shape)

temp = X

XX = np.reshape(temp, (temp.shape[0],(temp.shape[1]\*temp.shape[2]\*temp.shape[3])))

pca = PCA(n\_components = 180)

XX = pca.fit\_transform(XX)

print(XX.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(XX, Y, test\_size=0.2)

cls = KNeighborsClassifier()

cls.fit(X\_train, y\_train)

predict = cls.predict(X\_test)

acc = accuracy\_score(y\_test,predict)\*100

accuracy.append(acc)

text.insert(END,'KNN Prediction Accuracy : '+str(acc)+"\n")

Y1 = to\_categorical(Y)

cnn = Sequential()

cnn.add(Convolution2D(32, 3, 3, input\_shape = (64, 64, 3), activation = 'relu'))

cnn.add(MaxPooling2D(pool\_size = (2, 2)))

cnn.add(Convolution2D(32, 3, 3, activation = 'relu'))

cnn.add(MaxPooling2D(pool\_size = (2, 2)))

cnn.add(Flatten())

cnn.add(Dense(output\_dim = 128, activation = 'relu'))

cnn.add(Dense(output\_dim = 5, activation = 'softmax'))

cnn.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

print(cnn.summary())

cnn\_history = cnn.fit(X, Y1, batch\_size=16, epochs=10, validation\_split=0.2, shuffle=True, verbose=2)

cnn\_history = cnn\_history.history

cnn\_history = cnn\_history['accuracy']

acc = cnn\_history[9] \* 100

accuracy.append(acc)

text.insert(END,'CNN Prediction Accuracy : '+str(acc)+"\n\n")

bars = ('KNN Inference','CNN Inference')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, [accuracy[6],accuracy[7]])

plt.xticks(y\_pos, bars)

plt.show()

plt.title('KNN & CNN Inference Accuracy Performance Graph')

plt.show()

def predict():

filename = filedialog.askopenfilename(initialdir="testImages")

image = cv2.imread(filename)

img = cv2.resize(image, (64,64))

im2arr = np.array(img)

im2arr = im2arr.reshape(1,64,64,3)

img = np.asarray(im2arr)

img = img.astype('float32')

img = img/255

preds = classifier.predict(img)

predict = np.argmax(preds)

img = cv2.imread(filename)

img = cv2.resize(img, (800,400))

cv2.putText(img, 'Car Model Predicted as : '+names[predict], (10, 25), cv2.FONT\_HERSHEY\_SIMPLEX,0.7, (255, 0, 0), 2)

cv2.imshow('Car Model Predicted as : '+names[predict], img)

cv2.waitKey(0)

def graph():

bars = ('Linear Regression', 'KNN','SVM','CNN','SVM Inference','CNN Inference','KNN Inference','CNN Inference')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, accuracy)

plt.xticks(y\_pos, bars)

plt.show()

plt.title('All Algorithms Accuracy Performance Graph')

plt.show()

def close():

main.destroy()

font = ('times', 15, 'bold')

title = Label(main, text='Machine Learning-Based Car Model Prediction through Vehicle Pattern Recognition')

#title.config(bg='powder blue', fg='olive drab')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 13, 'bold')

ff = ('times', 12, 'bold')

uploadButton = Button(main, text="Upload Cars Dataset", command=uploadDataset)

uploadButton.place(x=20,y=100)

uploadButton.config(font=ff)

knnButton = Button(main, text="Run Linear Regression & KNN Algorithms", command=linearKNN)

knnButton.place(x=20,y=150)

knnButton.config(font=ff)

cnnButton = Button(main, text="Run SVM & CNN Algorithms", command=SVMCNN)

cnnButton.place(x=20,y=200)

cnnButton.config(font=ff)

svmButton = Button(main, text="Run KNN & SVM Algorithms", command=KNNSVM)

svmButton.place(x=20,y=250)

svmButton.config(font=ff)

kcnnButton = Button(main, text="Run KNN & CNN Algorithms", command=KNNCNN)

kcnnButton.place(x=20,y=300)

kcnnButton.config(font=ff)

predictButton = Button(main, text="Prediction Model", command=predict)

predictButton.place(x=20,y=350)

predictButton.config(font=ff)

graphButton = Button(main, text="Accuracy Comparison Graph", command=graph)

graphButton.place(x=20,y=400)

graphButton.config(font=ff)

font1 = ('times', 12, 'bold')

text=Text(main,height=30,width=85)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=450,y=100)

text.config(font=font1)

main.config()

main.mainloop()

**RESULTS AND DISCUSSION**

This project implements a graphical user interface (GUI) application that performs machine learning-based car model prediction using vehicle pattern recognition. It creates a user-friendly interface for loading a car dataset, running different machine learning algorithms, making predictions, and visualizing the results using various graphs. The machine learning models include linear regression, KNN, SVM, and CNN for car model prediction. Here's an explanation of the implementation:

* Importing Libraries:

Import the libraries like tkinter for creating the GUI, matplotlib for creating plots, numpy for numerical operations, pandas for data handling, sklearn for machine learning tasks, cv2 for image processing, keras for building neural networks, and others.

* GUI Initialization:

The main GUI window is created using tkinter. It sets the title and dimensions of the window.

* Global Variables:

Several global variables are declared to store information and data throughout the application, including filename, X, Y, model, X\_train, X\_test, y\_train, y\_test, and accuracy.

* Dataset Loading:

There is a function called uploadDataset() that allows the user to upload a dataset. The dataset is loaded from a directory specified by the user and is processed into X and Y data arrays. An image is also displayed to the user using OpenCV.

* Machine Learning Algorithms:

Linear Regression and K-Nearest Neighbors (KNN) are run together, and their accuracy is displayed.

Support Vector Machine (SVM) and Convolutional Neural Network (CNN) are run together, and their accuracy is displayed.

KNN and SVM are run together, and their accuracy is displayed.

KNN and CNN are run together, and their accuracy is displayed.

* Prediction:

There is a function called predict() that allows the user to select an image for car model prediction using the chosen machine learning model. The predicted car model is displayed using OpenCV.

* Graphs:

The code allows the user to view various accuracy comparison graphs. These graphs show the performance of different machine learning algorithms.

* GUI Components:

The code defines various GUI components, such as buttons for uploading the dataset, running algorithms, making predictions, and displaying graphs. Labels, text fields, and scrollbars are also used for displaying information and results.

* Event Handling:

The GUI components are associated with specific functions that are executed when the user interacts with them. For example, clicking the "Upload Cars Dataset" button triggers the uploadDataset() function.

* Main Loop:

The main.mainloop() function starts the main GUI loop, allowing the user to interact with the application.

**Results description**

Figure 1 is an illustration of the graphical user interface (GUI) application. It represents the initial state of the application, where the user interacts with the program to perform car model predictions. Figure 2 shows the GUI interface after the user has successfully uploaded a cars dataset for system training. It displays elements like buttons, text fields, and labels related to dataset loading. Figure 3 displays sample images of AM General SUV and Acura sedan model cars. These images are taken from the input dataset used to train the machine learning models. These images serve as examples of the data used for training. Figure 4 is a bar chart or graph that visually represents the comparison of accuracy between the linear regression and K-Nearest Neighbors (KNN) classification models. It shows the accuracy percentages achieved by each model, allowing users to see which one performs better.

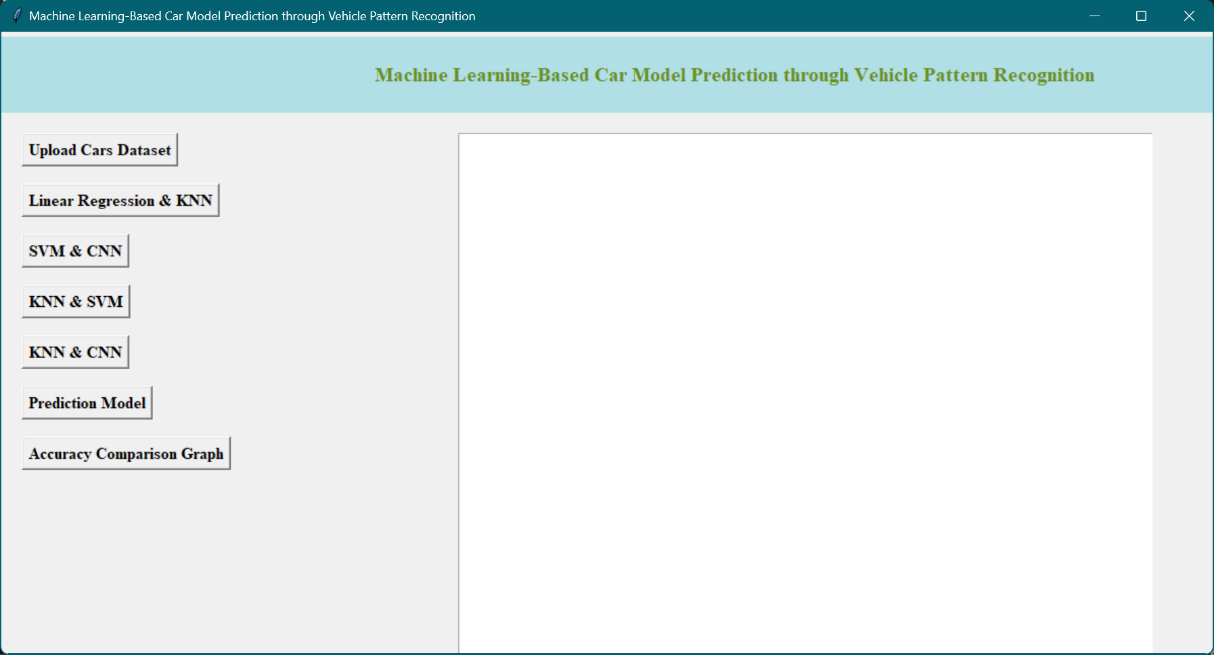


Figure 1: Illustration of GUI application for predicting the car model using the vehicle pattern recognition.

A screenshot of a computer

Description automatically generated

Figure 2: GUI application after uploading the cars dataset for system training.

A car driving on the road

Description automatically generated A silver car driving on the street

Description automatically generated A silver car with a white background

Description automatically generatedA blue car with a white background

Description automatically generated A white car parked in a parking lot

Description automatically generated A silver car parked in a parking lot

Description automatically generatedA car parked on the road

Description automatically generated The front of a car

Description automatically generated A blue car parked on a sidewalk

Description automatically generatedA car parked on grass near water

Description automatically generated The back of a silver car

Description automatically generated   

Figure 3: Sample AM general SUV, and Acura sedan model car images from the input dataset used for training.

A screenshot of a computer

Description automatically generated

Figure 4: Accuracy comparison of linear regression and KNN classification models.

Figure 5 shows a summary or architecture diagram of the proposed deep Convolutional Neural Network (CNN) model. It includes details about the layers, their configurations, and the flow of data through the network.

A screenshot of a computer program

Description automatically generated

Figure 5: Model summary of proposed deep CNN model.

Figure 6 illustrates the training performance of the deep CNN model over 10 epochs. It’s shown how the loss and accuracy change during training, allowing users to assess the model's convergence and performance.

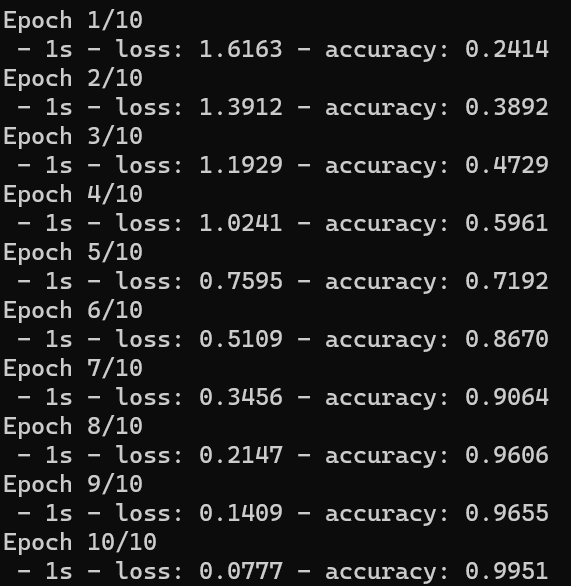


Figure 6: Training performance of deep CNN model for 10 epochs with loss and accuracy measures.

A screenshot of a computer

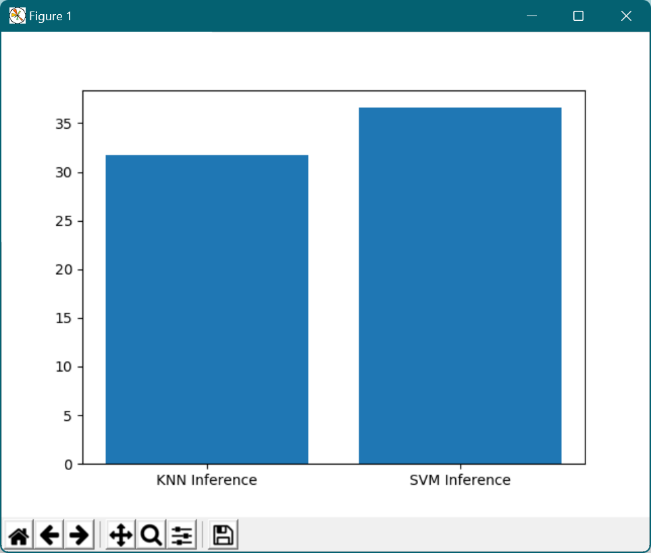
Description automatically generated 

Figure 7: Accuracy performance of KNN, SVM, and deep CNN models.

Figure 7 is a bar chart or graph that compares the accuracy performance of different models, including K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and deep CNN models. It provides a visual representation of how well each model performs. From the observations, deep CNN model outperforms all the other classifiers with 99.5% accuracy. Figure 8 shows the GUI application with accuracy results displayed on the interface. It includes text fields showing the accuracy scores achieved by the linear regression, KNN, SVM, and CNN models after training and testing the data.

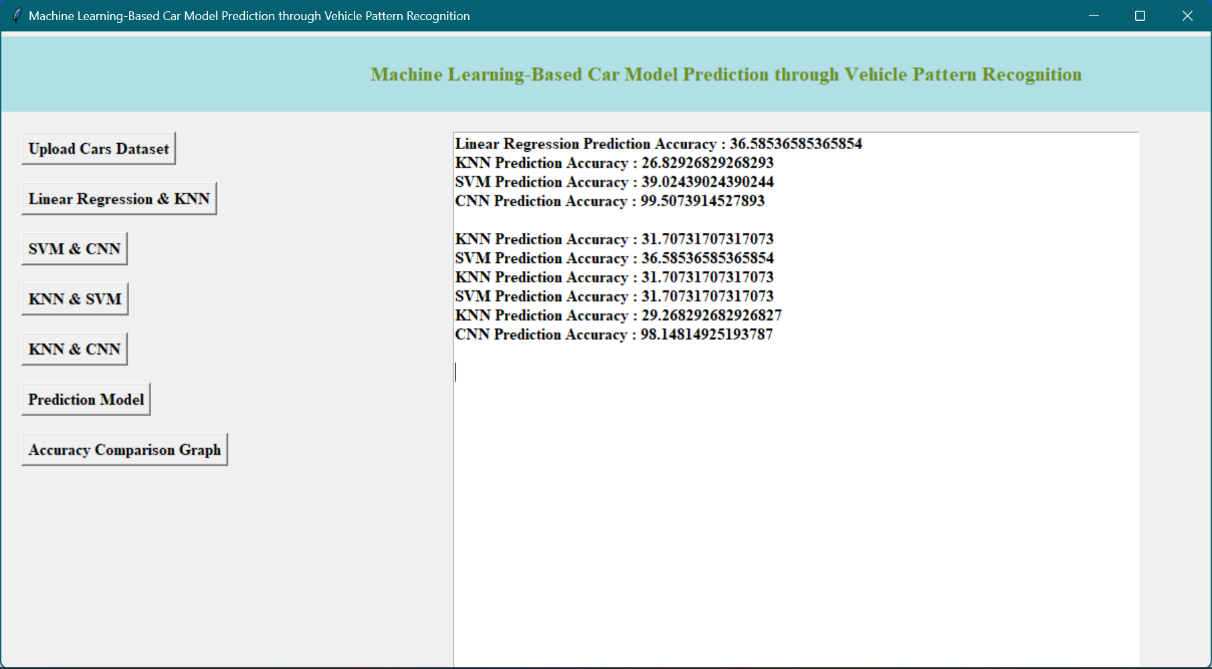


Figure 8: GUI application with the accuracies of linear regression, KNN, SVM, and CNN models after training and testing the data.

 A car parked in a driveway

Description automatically generated

A red car on display

Description automatically generated A back view of a car

Description automatically generated



Figure 9: Sample predictions on test data using proposed deep CNN model.

Figure 9 displays sample predictions made by the deep CNN model on test data. It shows input images and the corresponding predicted car models, allowing users to see how well the model performs in making predictions. From the obtained results, this project collectively provides a visual representation of the GUI application, training and testing results, model architectures, and performance comparisons, making it easier for users to understand and assess the machine learning-based car model prediction system.