CAR COLOR DETECTION MODEL

KANISHK KARAM

ABSTRACT

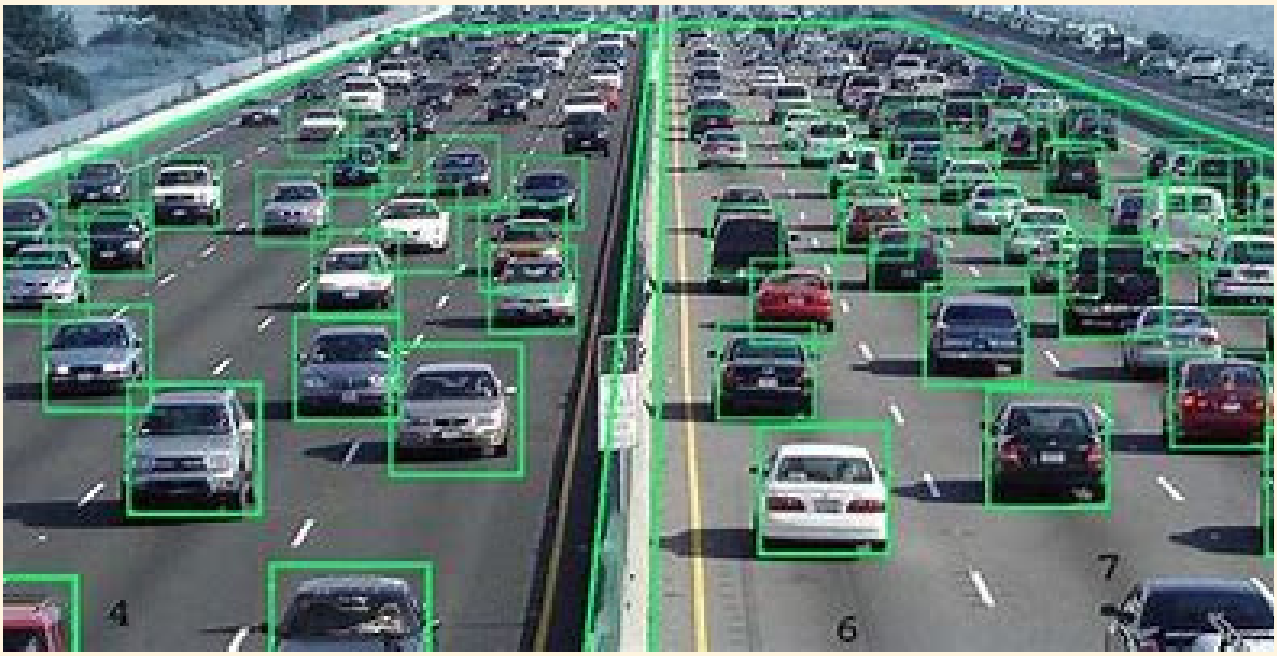
This project introduces a machine learning-based car color detection system to enhance traffic monitoring capabilities. The system classifies car colors in real-time and counts the number of cars at traffic signals. Unique visual markers, such as red rectangles for blue cars and blue rectangles for other cars, ensure clarity. Additionally, the model detects and counts people present at the signal. A user-friendly graphical interface (GUI) is developed to preview the input and processed output.

INTRODUCTION

- Problem Statement:
- Traffic management is a critical challenge in urban areas. Identifying vehicle attributes like color can assist in traffic analysis, incident management, and law enforcement. Similarly, tracking pedestrian presence enhances safety. This project bridges the gap by providing an automated solution for detecting car colors and pedestrians in traffic.
- Objectives:
- Develop a robust car color classification model using deep learning.
- Detect cars and pedestrians in video feeds using pre-trained object detection models.
- Create a GUI for intuitive use and real-time feedback.
- Applications:
- Traffic monitoring and incident detection.
- Pedestrian safety systems at traffic signals.
- Data collection for urban planning.

LITERATURE REVIEW

- Object Detection in Traffic Monitoring:
- Traditional systems rely on sensors or manual observation, which are costly and prone to errors. Computer vision offers a scalable and accurate alternative.
- Haar Cascades:
- Haar cascades are widely used for object detection due to their efficiency and simplicity. Pre-trained Haar cascades for vehicles and pedestrians are employed in this project.
- Deep Learning for Color Classification:
- CNNs (Convolutional Neural Networks) are state-of-the-art models for image classification tasks. Their ability to learn spatial hierarchies makes them ideal for tasks like car color detection.



METHODOLOGY

- Data Collection:
- The dataset is sourced from Kaggle, comprising labeled car images of various colors like black, blue, red, etc. Images are divided into training, validation, and testing subsets for model development and evaluation.
- Image Preprocessing:
- Resizing: All images are resized to 224×224224 \times 224224×224 to match the input dimensions of the CNN.
- Normalization: Pixel values are scaled to [0, 1] for faster convergence during training.
- Augmentation: Techniques like rotation, flipping, and brightness adjustments enhance model generalization.
- Model Training:
- Architecture: The CNN uses convolutional, pooling, and fully connected layers to extract features and classify images.
- Loss Function: Categorical cross-entropy is used to optimize predictions for multi-class classification.
- Evaluation Metrics: Accuracy, precision, recall, and F1 score.
- Car and Person Detection:
- Haar Cascades: Pre-trained cascades (haarcascade_car.xml and haarcascade_fullbody.xml) identify cars and people in video frames.
- Detection Flow:
    a. Convert frames to grayscale for detection.
    b. Extract regions of interest (ROI) for cars and classify their colors.
    c. Highlight detected cars and people using bounding rectangles.
- GUI Development:
- Design: The GUI, built using Tkinter, allows users to select video files or access real-time webcam feeds.
- Interactivity: Outputs are displayed with annotated bounding boxes and labels.

IMPLEMENTATION

- Key Modules:
  1. Color Classification: Predicts car color from detected ROIs using the trained model.
  2. Object Detection: Uses Haar cascades for car and person identification.
  3. GUI: Facilitates user interaction with options to upload videos or start live detection.
- Code Snippets:

Image Preprocessing for Model Input:

```
def prepare_image(image):
    image = cv2.resize(image, (224, 224))
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    return image
```

GUI for File Selection and Webcam Feed:

```
def open_file():
    file_path = filedialog.askopenfilename(filetypes=[("Video files", "*.mp4;*.avi;*.mov")])
    if file_path:
        process_feed(file_path)
```

Detection and Annotation:

```
for (x, y, w, h) in cars:
    car_roi = frame[y:y+h, x:x+w]
    prediction = model.predict(prepare_image(car_roi))
    car_color = class_labels[np.argmax(prediction)]
    rectangle_color = (0, 0, 255) if car_color == "blue" else (255, 0, 0)
    cv2.rectangle(frame, (x, y), (x+w, y+h), rectangle_color, 2)
```

RESULTS

- Performance Metrics:
- Training accuracy: 90%
- Validation accuracy: 88%
- Screenshots:
- Include annotated screenshots of detected cars and people with bounding rectangles and color labels.

CHALLENGES AND LIMITATIONS

- Challenges:
- Handling varying lighting conditions and occlusions in video feeds.
- Balancing model accuracy and real-time performance.
- Limitations:
- Haar cascades can sometimes miss objects in complex backgrounds.
- Color prediction accuracy might drop for low-resolution ROIs.

REFERENCES

- Kaggle Vehicle Color Recognition Dataset.
- OpenCV Documentation for Haar Cascades.
- Keras Documentation for Deep Learning Models.

CONCLUSION

This project successfully demonstrates an automated car color detection and pedestrian counting system. The integration of a machine learning model with real-time object detection provides an efficient solution for traffic monitoring. Future enhancements could include multi-camera setups and integration with traffic management systems.