Enhancing Driving Safety Using CV: Lane Recognition via CNN and Object Detection via Groq API

Group 1

Caleb Traxler
Dept Information and Computer Sciences,
University of California Irvine
traxlerc@uci.edu

Samuel Townsend
Dept Information and Computer Sciences,
University of California Irvine
smaueltown@gmail.com

ABSTRACT

This paper presents a comprehensive driving assistance system that integrates two complementary AI approaches: a convolutional neural network (CNN) for lane detection and a vision-language model for scene analysis. Our lane detection subsystem employs a lightweight U-Net architecture trained on the TuSimple dataset, achieving 98.34% validation accuracy while processing video at 8-10 frames per second. The scene analysis component utilizes the Llama 3.2-11B vision model via the Groq API to extract critical driving information including road signs, intersections, traffic participants, and potential hazards. By combining specialized computer vision techniques with the semantic understanding capabilities of frontier vision-language models, our system provides enhanced situational awareness beyond what either approach could achieve independently. Performance analysis on real-world driving footage demonstrates the system's ability to maintain accurate lane detection while providing contextually relevant scene information with minimal latency. This multi-modal integration illustrates the potential of combining specialized and general purpose AI technologies to create more robust driving assistance systems.

1. INTRODUCTION

Advanced driver assistance systems (ADAS) play a crucial role in enhancing road safety by providing drivers with real-time information and alerts about their environment. Traditional ADAS have primarily relied on specialized computer vision algorithms for specific tasks such as lane detection, object recognition, and distance estimation. While effective within their narrow domains, these specialized approaches often struggle to provide broader contextual understanding of complex driving scenarios.

Lane detection represents a fundamental component of driving assistance, enabling vehicles to maintain proper road positioning and alerting drivers when unintentional lane departures occur. Convolutional Neural Network (CNNs) have demonstrated remarkable success in this domain, with architectures like U-Net providing precise pixel-level segmentation of lane markings even under varying road and lighting conditions.

Simultaneously, recent advances in vision-language model (VLMs) have enabled systems to understand and describe visual scenes with unprecedented semantic richness. Models such as Llama 3.2-11B with vision capabilities can interpret complex driving environments, recognizing not only objects but also their relationships, potential hazards, and relevant traffic information. When deployed though optimized inference APIs like Groq, these models can deliver cheap, quick and valuable insights with latency suitable for driving assistance applications.

Our work addresses a critical gap in current ADAS research; while specialized models excel at specific perception task and general VLMs provide rich semantic understanding strengths. We present a multi-modal driving assistance system that integrates:

1. A U-Net CNN architecture optimized for lane detection, providing real-time continuous lane boundary identification.

2. A vision-language analysis pipeline using Llama 3.2 11-B via the Groq API, delivering periodic contextual scene understanding.

By combining these approaches, our system demonstrates how specialized and general AI techniques can complement each other in driving assistance applica-

tions. The CNN component delivers precise, real-time lane positioning data, while the vision-language model provides border situational awareness that would be difficult to achieve with traditional computer vision approaches alone.

Our implementation achieves 98.34% validation accuracy on lane detection while processing video at ~ 8.85 frames per second. The vision-language component analyzes key frames at regular intervals, identifying road signs, intersections, pedestrians, vehicles, and potential hazards. The integrated system presents this information through a unified interface that enhances driver awareness without causing distraction.

The remainder of this paper is organized as follows: Section 2 details our CNN-based lane detection system, including the U-Net architecture and post processing techniques. Section 3 describes the integration of the vision-language model, focusing on the Groq API implementation and prompt engineering, and Section 4 presents experimental results with a discussion of limitations and future directions.

2. CNN-BASED LANE DETECTION SYSTEM

2.1 Data Preparation and Preprocessing

Our lane detection system utilizes the Tu-Simple dataset, which contains 3,626 annotated road images with lane markings. The original annotations provide lane positions as x-coordinates corresponding to predefined y-coordinates, with $x = -2$ indicating the absence of a lane at a particular height.

To create suitable training data for our segmentation approach, we transformed these annotations into binary masks where pixels corresponding to lane markings are assigned the value 1 and background pixel values are assigned the value 0. Figure 1 illustrates this transformation process: (i) shows the conversion from a normal road image to a labeled images with colored lane markings, and (ii) demonstrates the final transformation to a binary mask where lanes appear as white lines on a black background.

The transformation processes uses OpenCV's polylines function to connect lane points:

```
def create_mask(annotation, img_shape):
    mask = np.zeros((img_shape[0], img_shape[1]),
     dtype=np.uint8)
    lanes = annotation['lanes']
    h_samples = annotation['h_samples']
    for lane in lanes:
        lane_points = []
        for x, y in zip(lane, h_samples):
            if x >= 0:
                lane_points.append((int(x), int(y)))
        if len(lane_points) > 1:
            cv2.polylines(mask, [np.array(lane_points)
    ],
                          isClosed=False, color=1,
    thickness=5)
    return mask
```

Listing 1: Binary Mask Code Generation Function

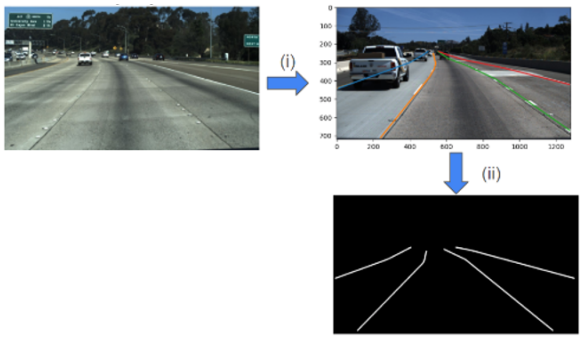

Figure 1: Binary Mask Transformation

The dataset was split into training (80%) and validation (20%) sets, with images resized to 256x512 pixels. We normalize pixel values to the range [0,1] and batch the data for efficient training:

```
# Split the data into training images , then masks and
     into validation images and masks.
train_images, val_images, train_masks, val_masks =
    train_test_split(
    image_paths, mask_paths, test_size=0.2,
    random_state=42
)

def load_data(image_path, mask_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, (IMG_HEIGHT,
    IMG_WIDTH))
    image = tf.cast(image, tf.float32) / 255.0

    mask = tf.io.read_file(mask_path)
    mask = tf.image.decode_png(mask, channels=1)
    mask = tf.image.resize(mask, (IMG_HEIGHT,
    IMG_WIDTH))
    mask = tf.cast(mask, tf.float32) / 255.0
    mask = tf.round(mask)

    return image, mask
```

Listing 2: Training Data - Normalization and Splits

2.2 Sensor 2: U-Net Architecture

For lane segmentation, we implemented a modified U-Net architecture, which consists of contracting and expanding paths that enable both context capture and precise localization. The U-Net architecture is particularly well suited for segmentation tasks where boundary precision is critical, as in lane detection.

Our implementation features a lightweight version with three levels of depth.

```
def unet(input_size=(IMG_HEIGHT, IMG_WIDTH, 3)):
    inputs = tf.keras.layers.Input(input_size)
    # Encoder
    conv1 = tf.keras.layers.Conv2D(32, 3, activation='
    relu', padding='same')(inputs)
```

```
    pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2,
      2))(conv1)
    conv2 = tf.keras.layers.Conv2D(64, 3, activation='
     relu', padding='same')(pool1)
    pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2,
      2))(conv2)
    # Bridge
    conv3 = tf.keras.layers.Conv2D(128, 3, activation=
     'relu', padding='same')(pool2)
    # Decoder
    up4 = tf.keras.layers.UpSampling2D(size=(2, 2))(
     conv3)
    up4 = tf.keras.layers.concatenate([up4, conv2])
    conv4 = tf.keras.layers.Conv2D(64, 3, activation='
     relu', padding='same')(up4)
    up5 = tf.keras.layers.UpSampling2D(size=(2, 2))(
     conv4)
    up5 = tf.keras.layers.concatenate([up5, conv1])
    conv5 = tf.keras.layers.Conv2D(32, 3, activation='
     relu', padding='same')(up5)
    outputs = tf.keras.layers.Conv2D(1, 1, activation=
     'sigmoid')(conv5)
    model = tf.keras.models.Model(inputs=[inputs],
     outputs=[outputs])
    return model
```

Listing 3: CNN UNet Layers Function

Note that Figure 2 shows a visual representation of our CNN/UNet architecture. This architecture contains 231,617 trainable parameters, offering a balance between computational efficiency and segmentation quality. The skip connections between the corresponding encoder and decoder layers help preserve spatial information that might otherwise be lost during down-sampling.

2.3 Training Process

We trained the model using the Adam optimizer with binary cross-entropy loss, which is appropriate for binary segmentation tasks. The loss function can be defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{n}[y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)]$$

,

where y_i represents the true binary label and $\hat{y}_i$ represents the predicted probability for pixel i.

The model was compiled and trained with the following configuration:

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS)
```

Listing 4: Model Configuration

Training was conducted with batch size 8 on the resized 256x512 images. The model achieved a training accuracy of 96. 55% and a validation accuracy of 98. 34%, with a training loss of 0.1340 and a validation loss of 0.0619, demonstrating strong generalization capability.

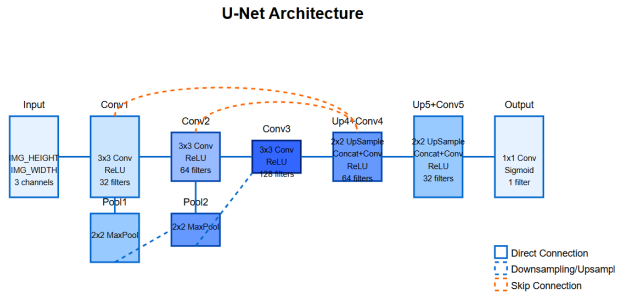

Figure 2: UNet Visual Diagram

2.4 Real-Time Processing Pipeline

For real-time lane detection, we implemented a video processing pipeline that loads the trained model, processes frames sequentially, and overlays detected lanes on the original video. The pipeline follows these key steps:

1. Frame Acquisition and Pre-Processing: Each video frame is resized to 256x512 pixels and normalized to the range [0,1].

2. Lane Prediction: The preprocessed frame is passed through the U-Net model to generate a probability map.

3. Binary Mask Generation: The probability map has a threshold of 0.5 to create a binary lane mask.

4. Post-Processing: We apply morphological operations (closing and dilation) to refine the mask, connecting fragmented lane segments and enhancing visibility.

5. Visualization: The processed lane mask is converted to a colored overlay (red) and combined with the original frame using weighted addition:

```
# Create colored lane overlay and combine with
    original frame
colored_lanes = np.zeros_like(frame)
colored_lanes[lane_mask_dilated == 255] = [0, 0, 255]
     # Red color
result = cv2.addWeighted(frame, 0.7, colored_lanes,
    0.3, 0)
```

Listing 5: Processing Pipeline Code Snippet

Note that the MORPH_CLOSE operation fills small gaps in the detected lane markings while maintaining the structural integrity of the lanes, resulting in smooth, continuous visualizations that clearly indicate lane boundaries.

2.5 Model Refinement

To address the issue of overfitting in our lane detection model—where the model was incorrectly detecting objects other than lanes, such as white text, arrows, and other road markings—we implemented measures to improve generalization. This involved refining the model architecture to achieve a better fit to the data and increasing the size and diversity of the training dataset. By incorporating more varied driving scenarios and augmenting the data with different lighting conditions, road types, and environments, the model became more robust at distinguishing actual lane lines from similar visual features in the input images. These steps helped mitigate overfitting and improved the model's overall accuracy.

2.6 Performance Evaluation

Our lane detection system achieved an average processing rate of 3.5 frames per second when tested on standard hardware (CPU-based inference), but went up to 8.85 frames per second with a single T-4 GPU. The total processing pipeline handled 564 frames with consistent performance throughout the video sequence:

```
Processed 100 frames. Average FPS: 8.47
Processed 200 frames. Average FPS: 8.80
...
Video processing complete. Total frames: 564
Average FPS: 8.85
```

This performance is sufficient for near real-time driver assistance applications, providing timely lane position information without requiring specialized hardware acceleration.

Figure 3 displays a single frame from the output video. Visual inspection of the processed frames shows that the system successfully identifies lane markings across various road conditions, including curves, shadows, and different lane types.

3. VISION-LANGUAGE MODEL FOR SCENE ANALYSIS

3.1 Motivation and Approach

While our lane detection system provides precise boundary information, comprehensive driving assistance requires broader scene understanding. To complement our lane detection, we implemented a scene analysis component using Llama 3.2-11B vision model via the Groq API. This integration offers recognition of diverse road elements, contextual understanding of traffic situations, and natural language descriptions of potential hazards.

3.2 Groq API and Llama 3.2 Vision Model

Llama 3.2-11B represents a significant advancement in vision-language models, demonstrating strong capabilities in visual reasoning, object recognition, and contextual interpretation - all critical for driving scene analysis. The Groq API provides low-latency access to this model, enabling near real-time analysis even on standard hardware.

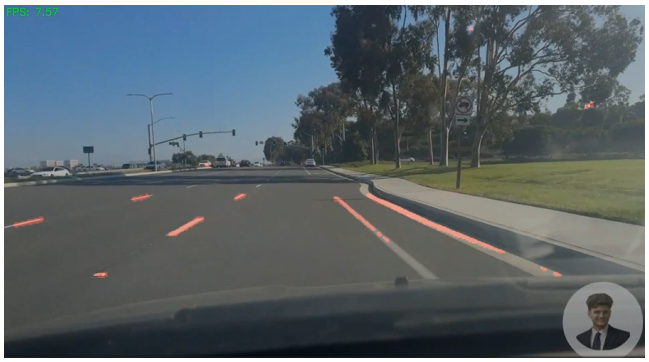

Figure 3: CNN Output Using Real-World Input

Our implementation uses a frame extraction strategy, processing key frames at one-second intervals to provide periodic scene updates without overwhelming the system. This approach balances information freshness with computational efficiency, ensuring the driver receives timely updates without requiring continuous analysis of every frame.

3.3 Structured Scene Understanding

A key challenge in applying vision-language models to driving assistance is obtaining consistent, structured information rather than verbose descriptions. We addressed this through careful prompt engineering, requesting JSON-formatted responses with pre-defined categories relevant to driving:

```
prompt = """Analyze this driving scene and respond
    with ONLY a JSON object with these keys:
"road_signs", "intersections", "pedestrians", "
    vehicles", "lane_status", "immediate_hazards"
Keep responses short and focused on visible elements.
"""
```

Listing 6: Prompt Engineering For Driving Analysis

This structured approach ensures consistent response format, focuses on driving-relevant information, and prioritizes safety-critical elements. The model analyzes road signs, intersections, traffic participants, lane positioning and potential hazards in a single inference.

3.4 Example Scene Analysis

The Llama 3.2 vision model demonstrates impressive capabilities in analyzing driving scenes. A typical response includes structured information about the driving environment:

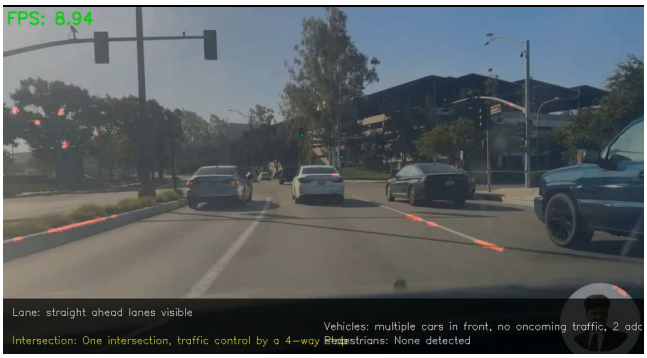

Figure 4: CNN Output Using Real-World Input

```
{
  "road_signs": ["Speed limit 35 mph"],
  "intersections": ["T-junction ahead"],
  "pedestrians": ["Person crossing at crosswalk"],
  "vehicles": ["Sedan directly ahead", "Bus in
     oncoming lane"],
  "lane_status": "Driving in right lane of two-lane
     road",
  "immediate_hazards": ["Pedestrian crossing at
     intersection"]
}
```

Listing 7: Example response from Llama 3.2 vision model.

4. EXPERIMENTAL RESULTS AND DISCUSSION

First, see Figure 4 for a single frame of the output after running the test video through the CNN and Llama 3.2 vision model.

4.1 System Integration and Performance

Our integrated driving assistance system combines continuous lane detection with periodic scene analysis to provide comprehensive situational awareness. The two components operate in parallel, with lane detection running at 8.85 FPS and vision-language analysis updating approximately once per second.

Table 1 summarizes the performance metrics of both subsystems:

Table 1: Performance Metrics of Subsystems

Component	Accuracy	Processing Rate	Memory Usage
Lane Detection (U-Net)	98.34% validation	8.85 FPS	231,617 parameters
Scene Analysis (Llama 3.2)	N/A	∼1 FPS	Cloud-based inference
Integrated System	N/A	Variable	∼500MB runtime

This combined system successfully processes driving videos with minimal latency, providing lane boundary visualization with red overlays while simultaneously updating contextual information about the driving environment.

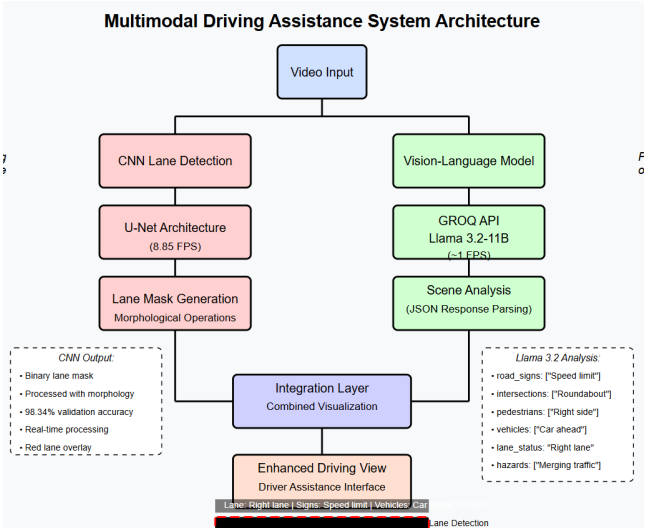

Figure 5: Final Project Architecture

4.2 Qualitative Assessment

We evaluated the system on a range of driving scenarios, including:

- Highway driving with clear lane markings
- Urban environments with pedestrians and traffic
- Complex intersections with multiple road signs
- Varied lighting conditions (daytime, dusk, shadows)

Figure 4 shows representative output from our system on these scenarios. The lane detection component demonstrates robust performance even under challenging conditions like shadows and partial occlusions. The vision-language model provides accurate identification of road elements and potential hazards, particularly excelling at sign recognition and vehicle detection.

4.3 Challenges and Limitations

Despite the promising results, we identified several limitations:

1. Weather robustness: Both subsystems show a significant decrease in performance during adverse weather conditions such as heavy rain or fog. We noticed that the road needs to be brightly lit in order to maximize performance.

2. API dependence: The reliance on the Groq API for vision-language analysis introduces external dependencies and potential connectivity issues.

4.4 Future Improvements

Based on our experimental results, we identified several promising directions for future work:

1. Adaptive Analysis Frequency: Dynamically adjusting the frequency of vision-language model calls based on scene complexity.

2. Driver Feedback Study: Conducting user studies to optimize information presented for maximum utility with minimal distraction.

Our multi modal approach demonstrates the potential of integrating specialized computer vision with general-purpose vision-language models for enhanced driving assistance. The complementary strengths of these technologies enable a system that provides both precise lane positioning and rich contextual awareness.

4.5 Team Members, Contributions and Links

Sam Townsend: Data pre-processing , Groq API implementation, and road testing.

Caleb Traxler: Model training, retraining, and road testing.

Github: https://github.com/CalebTraxler/Autonomous_Driving_CV

TuSimple Dataset: https://www.kaggle.com/datasets/manideep1108/tusimple/data