1. Problem Statement:

Car lane detection is crucial for both advanced driver assistance systems (ADAS) and autonomous vehicles for safety and navigation. It helps vehicles stay within their lane, preventing accidents and ensuring adherence to traffic rules. By identifying the position and shape of the lane, the vehicle can avoid drifting or colliding with other vehicles, especially due to driver distraction or fatigue.

1. Advantages of Car Lane Detection:

* Safety: Accurate lane detection helps prevent lane departures, which can lead to accidents, especially in situations where drivers are fatigued or distracted.
* Autonomous Driving: For self-driving cars, lane detection is a fundamental component for navigation and path planning, ensuring the vehicle stays within its designated lane.
* Real-time Vision: Lane detection algorithms must be robust and real-time to work effectively in various conditions, including challenging lighting and lane marking scenarios.

1. Dataset Loading:
   * The libraries I used for handling files are: (os, json).
   * Firstly, I downloaded the *TUSimple* dataset from *Kaggle.*
   * Then, I parsed the JSON files that contain:

*lanes*: lists of x-coordinates per lane line.

*h\_samples*: list of all y-coordinates across all lanes.

*raw\_file*: path to the image.

1. Preprocessing:

* The libraries I used for image processing are: (cv2, NumPy)
* Plotting using (matplotlib)
* In the *LaneDataset* class, I load the images then convert them to RGB, resize to (128,256) to normalize all images to a standard size.
* I used *img = torch.tensor* to convert the image from NumPy array to PyTorch Tensor and change data type to float, which is necessary for image processing in PyTorch, as many operations expect floating-point inputs.
* *.permute(2, 0, 1)* to rearrange the dimensions(Height, Width, Channels) to standard format(C, H, W) in PyTorch of the tensor.
* Apply Z-normalization*= (img-mean)/std.*
* Semantic segmentation:
* Is a computer vision task where each pixel in an image is assigned a class label
* A segmentation mask is an image (same width/height as the input image) where:
* Each pixel's value = 1 if it belongs to a lane. Pixel value = 0 if it’s background.
* This way, your model learns which pixels represent lanes.
* So, I convert lane coordinates to segmentation mask using *cv2.circle* to draw lane points into the mask.

1. Dataloaders:

* *train\_loader = DataLoader:* wraps the LaneDataset with batching, shuffling, and multiprocessing.
* Batches are size 16.

1. Model Architectures:
2. U-Net:

* Encoder-decoder architecture with skip connections.
* Downsampling: Conv → ReLU → Conv → ReLU → MaxPool.
* Upsampling: TransposeConv → Concat → Conv.
* Final output: 2-channel segmentation map (for 2 classes: lane vs background).
* Loss=0.15 after 10 epochs.
* Accuracy= 0.944, Precision= 0.409, F1= 0.568, IoU= 0.396

1. FastSCNN (lightweight):

* Lightweight encoder-decoder.
* Encoder: Two downsampling convs.
* Decoder: Two upsampling transpose convolutions.
* Faster and more efficient for real-time.
* Loss= 0.395 after 10 epochs.
* Accuracy=0.909, Precision= 0.236, F1= 0.334, IoU=0.2

1. Enet:

* Ultra-lightweight model for fast inference.
* Fewer filters and aggressive downsampling.
* Loss= 0.388 after 10 epochs.
* Accuracy=0.904, Percision= 0.229, F1= 0.33, IoU=0.198

**Therefore, the highest Accuracy model with lowest Loss is U-Net.**

1. Training & Evaluation:

* Split the dataset to train\_data=80% and validation\_data=20%

*train\_and\_evaluate* function*:*

* Moves model to GPU.
* Uses CrossEntropyLoss with class weights to handle class imbalance (lane = 1.0, background = 0.1`).
* Optimizer: Adam (learning rate = 0.001)
* Trains for 10 epochs.
* Evaluates on val\_loader using:
* Accuracy: Fraction of correct predictions.
* Precision: Focuses on correct positives (how precise are my lane predictions).
* F1 Score: Harmonic mean of precision and recall.
* IoU
* Confusion Matrix
* Saves loss plot and model weights.

1. Visualization:

* Trains and evaluates each model (UNet, FastSCNN, ENet).
* For 2 random validation images, displays:
* Original image
* Ground truth mask
* Predicted mask
* Saves visualizations as PNGs.