Z-ENG:Line association using neural network

GitHub repository link: https://github.com/DeliaJing/Line-association-using-neural-network.git

Task description:

This project implements a line association model for lane detection across consecutive video frames. The goal is to identify which lane lines in one frame t correspond to those in the next t+1, enabling the tracking of lanes over time.

The system compares two approaches for line matching:
1. Rule-based method using geometric heuristics (e.g., angle, center distance, length).
2. Neural network-based method trained to predict match probabilities from extracted features.

The program supports three main modes:
1. Training: Learns a line-matching model from training data.
2. Visualization: Displays matched lane lines between frames using either method.
3. Evaluation: Measures agreement and runtime between the rule-based and neural models.

Dataset: CULane Dateset

I chose the CULane dataset for the following reasons:

1.Consecutive frame pairs with annotations:
CULane provides raw images along with .lines.txt annotation files across consecutive video frames. This makes it an ideal choice for line association tasks, as the annotated lane lines can be directly extracted and used for training and visualization.

2.Predefined train/val/test splits:
The dataset includes official train.txt, val.txt, and test.txt lists, each containing image paths and their corresponding annotation files. This makes it easy to generate training, validation, and testing sets for the line association model without additional preprocessing.

3.Diverse road environments and sufficient scale:
CULane covers a wide range of driving scenarios, including highways, city roads, night scenes, and challenging lighting conditions. Its large scale ensures that the neural network model can learn stable, generalizable features for lane matching.

Neural Network Model:Feedforward neural network

This project uses a simple feedforward neural network (also known as a Multi-Layer Perceptron, or MLP) for the task of lane line association between consecutive video frames.

What the Model Does

The model takes a 7-dimensional feature vector that represents the relationship between two lane lines (one from frame t, and one from frame t+1). It outputs a probability between 0 and 1, indicating how likely these two lines are a match.

The feature vector includes:
Cosine similarity of line direction
Distance between line centers
Line length difference
Coordinate differences of line endpoints

The model is trained using binary cross-entropy loss, where matched pairs are labeled as 1 and non-matched pairs as 0

Why This Model Was Chosen

1).Simple and efficient
A feedforward network is fast to train and computationally lightweight, making it suitable even for limited hardware.

2).Sufficient for low-dimensional features
Since the input feature vector has only 7 dimensions, more complex models (like CNNs or transformers) are unnecessary.

3).Interpretable and easy to extend
The model is easy to understand and can be scaled up by adding more layers or features in future versions.

Model Architecture in Code

The architecture implemented in LineAssociationModel(nn.Module) consists of:

1).Input Layer:
A fully connected layer that takes in 7 features.

2).Two Hidden Layers:
Each followed by ReLU activation functions. The number of hidden units is set to 64 by default but can be adjusted.

3).Output Layer:

A single neuron output passed through a Sigmoid function to produce a probability.

```
self.model = nn.Sequential(
    nn.Linear(input_dim, hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, 1),
    nn.Sigmoid()
)
```

Technology Stack

This project was implemented using the following technologies:

Python 3.10

Used as the primary programming language due to its strong ecosystem for deep learning, data processing, and visualization.

PyTorch 2.0:

Chosen for building and training the neural network model (LineAssociationModel). PyTorch provides flexibility in model design, efficient GPU support, and integration with tools like DataLoader for batch processing.

OpenCV 4.8:

Used for visualizing lane lines and drawing matched pairs across consecutive frames. It offers fast image manipulation and integration with NumPy arrays.

Matplotlib 3.7:

Used for displaying side-by-side frame comparisons in a more readable format for analysis and debugging.

scikit-learn 1.2

Used for evaluating model performance (e.g., accuracy_score) during neural vs. rule-based comparison.

Project Realization

This project successfully fulfills all the requirements of the assignment. A complete neural-network-based line association system was implemented in Python. The project includes:

1.Designing an input feature structure for comparing lane lines across frames

2.Selecting and implementing a suitable neural network model for line matching
3.Generating the full training pipeline, including the creation of training, validation, and test sets from the CULane dataset
4.Training the model using PyTorch and evaluating it based on both accuracy and runtime performance
5.Comparing the neural network’ s predictions with a classical rule-based solution to assess matching agreement

All components — including data preprocessing, model training, evaluation, and visualization — were developed from scratch and tested successfully on the CULane dataset.

User guide:

1. Test Purpose

If you do not want to download the full CULane dataset (since it is quite large), I have provided a sample version instead. You can find it at the following path in my github repositoy:

...\Line association using neural network\testSample\CULaneDataset

After you find the path,then substitue the base_dir path.

```
base_dir = r"C:\Users\DELL\Documents\CULaneDataSet"
```

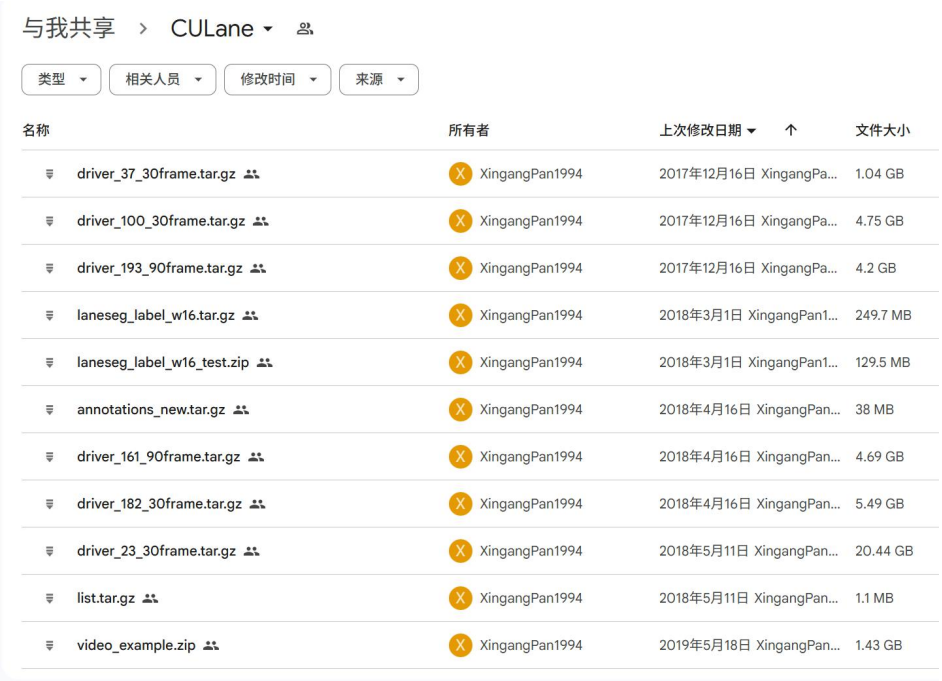
2. Train line association model from scrach:

1).Download the CULane dataset

Visit the official CULane dataset link:
https://drive.google.com/drive/folders/1mSLgwVTiaUMAb4AVOWwlCD5JcWdrwpvu

2).Verify the dataset

After opening the link, make sure the file structure matches the screenshot below.If it matches, proceed to download the entire dataset.



3).Set up the data path

In your code, locate the base_dir variable and update it to the full path where the CULane dataset is located on your machine.

```
base_dir = r"C:\Users\DELL\Documents\CULaneDataSet"
```

4) .Specify the model path

Set the model_path variable to your preferred filename.The trained model will be saved to this path after training is completed.

```
model_path = os.path.join(base_dir, "association_model_final_train.pth")
```

Once these steps are done, you can run the script with train = True to start training the neural line association model from scratch.

3. Use a Pretrained Line Association Model (Provided)

If you prefer not to train the model from scratch, you can use the pretrained model file that is included/provided with the project. Follow the steps below to set it up :

1).Place the pretrained model

Copy the provided model file (e.g., association_model_final.pth) to your desired directory.

2).Update the model path in the code

Open your script and set the model_path variable to the full path of the model file you just placed.

```
model_path = os.path.join(base_dir, "association_model_final_train.pth")
```

3) Set execution flags

In the if __name__ == "__main__": block, make sure:train = False

4. Evaluate Line Association model

To evaluate and compare the performance between the classic rule-based method and the neural network model, set:evaluate = True

Please note that evaluation requires a trained model. You must have either:Just trained the neural model (train = True), or valid pretrained model file at the path specified by model_path

When it trigger the evaluation pipeline, it will show report inside terminal as following:

```
Evaluating

Checked 100 image pairs.
Neural network made the same match as the classic method 94.95% of the time.
Classic method took about 0.0002 seconds per image.
Neural network took about 0.0026 seconds per image.
```

5. Visualize Lines matches between consecutive frames

This project supports visualizing matched lane lines between consecutive video frames using either the classic rule-based method or the neural network model.

1) . To visualize classic matches:Simply set the model to None

```
visualize_prediction(test_loader, None)
```

2) To visualize neural network matches:
   You must either have just trained the model (train = True), or provide a pretrained model file and set the correct model_path.

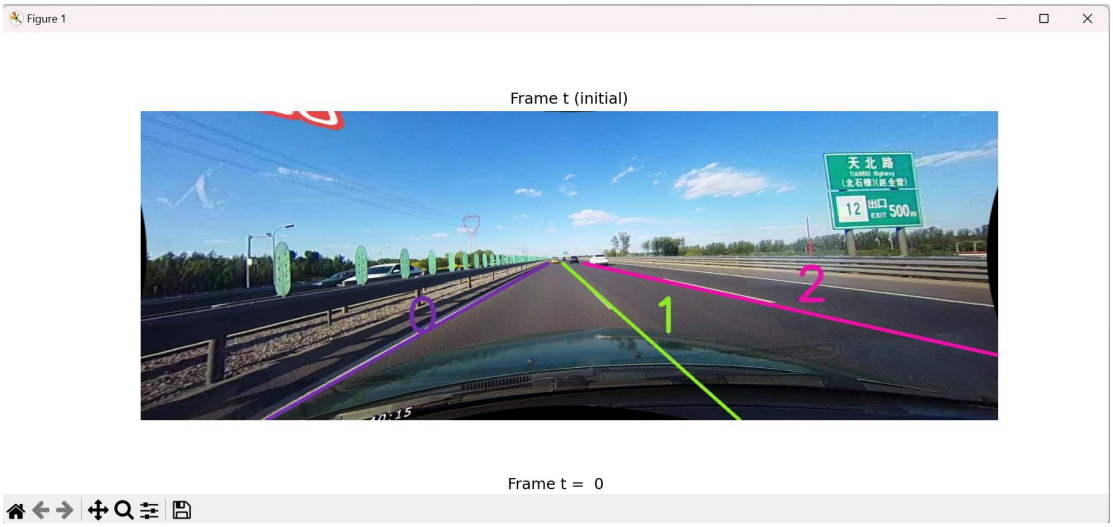
```
visualize_prediction(test_loader, model)
```

Visual results are displayed using matplotlib, showing two consecutive frames:

Frame t (previous)
Frame t+1 (current)

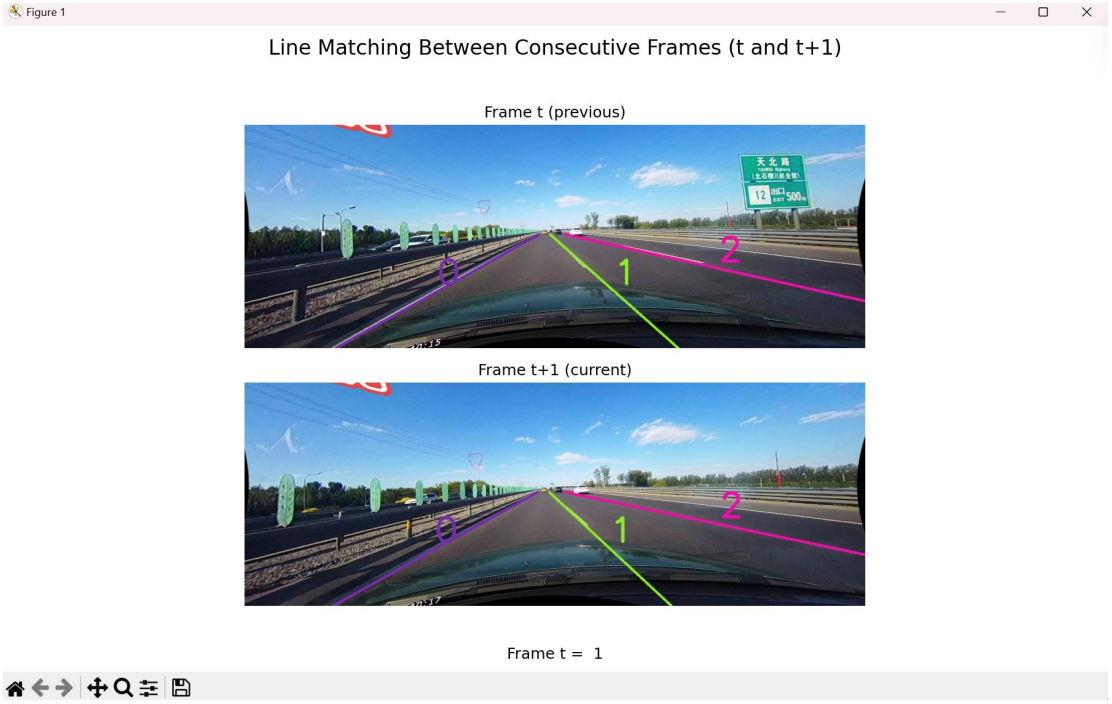
Matched line pairs are drawn in the same color and labeled with a consistent ID, allowing easy visual tracking across frames.

On the first frame, no previous lines exist to match — the system will only initialize and assign IDs to each line in the first frame.



To view the next pair of frames, you must close the current figure window (click the ✕ "X" in the top corner).

Once the current figure is closed, the next visualization will automatically be displayed.



If you do not want to continue visualize more pairs of frame,you can stop the program earlier in the terminal press:ctrl + c to stop the program.

Function and Class Descriptions

parse_lines_txt_np(lines_txt_path)

This function parses a .lines.txt file where each line contains a sequence of coordinates (x1 y1 x2 y2 ...). It converts each line into a NumPy array of shape (N, 2), representing one lane line.

The function is useful for preprocessing lane detection datasets.

Inputs:
lines_txt_path (str): Path to the .lines.txt file. Each line should contain an even number of float or integer values representing alternating x and y coordinates.

Outputs:
List[np.ndarray]: A list of NumPy arrays. Each array represents a single lane line, shaped as (N, 2) with integer x and y coordinates.

get_lines_endPoints(lane_lines_np)
This function extracts the starting and ending points of each lane line from a list of NumPy arrays. Each array represents one lane line, consisting of multiple (x, y) coordinate pairs.

Inputs:
lane_lines_np (List[np.ndarray]): A list of NumPy arrays, each of shape (N, 2), representing individual lane lines as sequences of (x, y) coordinates.

Outputs:
List[List[int]]: A list of endpoints in the format [x1, y1, x2, y2] for each lane line, where (x1, y1) is the first point and (x2, y2) is the last

CULaneLineTxtDataset
This class implements a PyTorch Dataset for the CULane dataset, which loads images and their associated .lines.txt lane annotations. It outputs the RGB image and corresponding lane line endpoints in tensor form.

Inputs:
1. base_dir (str): Path to the root directory that contains images and annotation files.
2. image_list_path (str): Path to a text file listing image paths (relative to base_dir).
Each line corresponds to one .jpg image and a matching .lines.txt file.

Outputs:
1. image_tensor (torch.Tensor): A normalized RGB image tensor of shape [3, H, W].
2. endPoints_tensor (torch.Tensor): A tensor of shape [N, 4], where each row is [x1, y1, x2, y2] representing the first and last points of a lane line.

association_lines(previous_lines, current_lines, angle_thresh=0.88, center_dist_thresh=70, endpoint_dist_thresh=60, length_ratio_thresh=0.5)
This function performs line association between two consecutive frames (e.g., in a video) by comparing geometric features such as direction, position, and length. A greedy matching algorithm is used to find non-overlapping matches with the highest compatibility scores.

Inputs:
1. previous_lines (List[List[float]]): Line segments from the previous frame, in the form [x1, y1,

x2, y2]. Can also be nested [[x1, y1], [x2, y2]].

2. current_lines (List[List[float]]): Line segments from the current frame in the same format.
3. angle_thresh (float): Threshold for cosine similarity between line directions (default: 0.88).
4. center_dist_thresh (float): Maximum distance allowed between line centers (default: 70 pixels).
5. endpoint_dist_thresh (float): Maximum distance between corresponding endpoints (default: 60 pixels).
6. length_ratio_thresh (float): Minimum ratio between the shorter and longer line lengths (default: 0.5).

Outputs:

List[Tuple[int, int]]:

A list of tuples (prev_idx, curr_idx) representing matched lines between the previous and current frames.

extract_features(previous_line, current_line)

This function computes a set of geometric features between two line segments. It is typically used for evaluating how similar two lines are, such as in tracking or temporal association between frames in a video.

Inputs:

1. previous_line (List[float]): The first line segment defined as [x1, y1, x2, y2], typically from the previous frame.
2. current_line (List[float]): The second line segment in the same format, typically from the current frame.

Outputs:

List[float]:

A list of 7 features:

1. Cosine similarity of direction vectors
2. Distance between line centers
3. Absolute difference in lengths
4. Δ x of start point
5. Δ y of start point
6. Δ x of end point
7. Δ y of end point

extract_line_features(previous_lines, current_lines, association_model=None)

Generates feature vectors for all possible line pairs between two frames, along with binary labels indicating match or non-match.

Inputs:

1. previous_lines (List[List[float]]): Line segments from the previous frame.

2. current_lines (List[List[float]]): Line segments from the current frame.
3. association_model (optional): A neural network used to predict matching pairs. If None, a 4. rule-based method is used.

Outputs:
1. features (List[List[float]]): A list of feature vectors (e.g., 7 values each) for every pair.
2. labels (List[int]): A list of binary labels (1 if matched, 0 otherwise).

collect_association_features(lines_Dataloader, association_model=None)
Collect features and labels from a dataloader for line association training.Used to generate training or validation data by extracting geometric features between lines across consecutive frames, with labels indicating matches.

Inputs:
1. lines_Dataloader: A PyTorch-like data loader that yields (image_batch, line_batch), where:
2. image_batch: Tensor of shape [B, 3, H, W]
3. line_batch: Tensor of shape [B, N, 4]
4. association_model (optional): A neural model to predict matches. If not provided, uses rule-based matching.

Outputs:
1. features (List[List[float]]): Feature vectors for all previous-current line pairs.
2. labels (List[int]): Binary labels (1 = matched, 0 = not matched).

LineAssociationModel(input_dim=7, hidden_dim=64)
A fully connected neural network model that predicts the probability that two lines from different frames correspond to the same physical lane.

Architecture:
Input: 7-dimensional feature vector per line pair
Hidden layers: Two fully connected layers with ReLU activations
Output: A sigmoid probability representing the match likelihood

Inputs:
input_dim (int): Number of input features (default: 7)
hidden_dim (int): Hidden layer size (default: 64)

Forward input:
x: A tensor of shape [B, input_dim], where B is the batch size.

Output:
A tensor of shape [B] with match probabilities for each input pair.

Use case:

Used in training or inference to learn line-pair similarity from features such as angle, center distance, and length difference.

validate_model(model, val_loader, criterion)
Evaluates a binary classification model on a validation dataset. Returns average loss and accuracy.

Inputs:
1. model (nn.Module): The PyTorch model to evaluate (e.g., a trained line association network).
2. val_loader (DataLoader): A PyTorch DataLoader that provides (features, labels) batches from the validation set.
3. criterion: The loss function used to compute prediction error (e.g., nn.BCELoss()).

Outputs:
1. avg_loss (float):The average loss over all validation batches.
2. accuracy (float):The classification accuracy (based on a threshold of 0.5 for predicted probabilities).

train_association_model(association_train_loader, val_loader, num_epochs=10, patience=5)
Trains a LineAssociationModel using binary classification (matching vs. non-matching lines). The model is trained with early stopping to avoid overfitting.

Inputs:
1. association_train_loader (DataLoader): PyTorch DataLoader providing training features and labels.
2. val_loader (DataLoader): PyTorch DataLoader for validation evaluation.
3. num_epochs (int): Maximum number of training epochs (default: 10).
4. patience (int): Number of consecutive epochs without improvement before early stopping (default: 5).

Training Details:
1. Loss: BCELoss (Binary Cross Entropy)
2. Optimizer: Adam (lr=1e-3)
3. Evaluation: Tracks validation loss and accuracy
4. Early stopping: Stops if validation loss doesn’t improve for patience epochs

Output:
association_model (LineAssociationModel): The best model (based on lowest validation loss) with weights restored.

predict_matches_nn(previous_lines, current_lines, association_model, threshold=0.5)
Uses a trained neural network to predict the best matches between lines from consecutive frames based on learned geometric features.

Inputs:

1. previous_lines (List[List[float]]): List of lines from the previous frame in [x1, y1, x2, y2] format.
2. current_lines (List[List[float]]): List of lines from the current frame in the same format.
3. association_model (nn.Module): A trained PyTorch model that outputs a match probability.
4. threshold (float, optional): The minimum probability required to accept a match (default: 0.5).

Output:
matches (List[Tuple[int, int]]): List of (prev_idx, curr_idx) index pairs representing matched lines. If no suitable match is found, curr_idx is -1.

Notes:
Each line in current_lines is matched at most once.
The model is evaluated in eval() mode and inference uses torch.no_grad().

draw_matched_lines(previous_image, previous_lines, current_image, current_lines, matches, matched_lines_details_previous, match_id_dict, frame_idx)
Visualizes matched lane lines between two consecutive frames with consistent, color-coded IDs. Useful for debugging and presentation of line tracking logic.

Inputs:
1. previous_image (np.ndarray): The previous frame's image.
2. previous_lines (List[List[int]]): List of lines from the previous frame, each in [x1, y1, x2, y2] format.
3. current_image (np.ndarray): The current frame's image.
4. current_lines (List[List[int]]): List of lines from the current frame.
5. matches (List[Tuple[int, int]]): Index pairs of matched lines between frames.
6. matched_lines_details_previous (List[List]): Tracks [line_idx, color, id] for matched lines in the previous frame.Updated in-place with info for the current frame.
7. match_id_dict (dict): Dictionary mapping matched index pairs to consistent IDs.
8. frame_idx (int):The index of the current frame (used for labeling or display).

Output:
No return value. The function displays side-by-side annotated frames with matched lines and IDs.

Handles:
1. ID initialization on first frame
2. Color reuse for consistent visual tracking
3. Line drawing and labeling using cv2 and matplotlib

show_matched_frames(img1, img2, frame_idx)
Displays two consecutive frames (frame t and t+1) side-by-side to visualize line matching results, including annotations and frame index.Called after line matching and drawing, this function helps visually verify the correctness of line association across frames.

Inputs:

1. img1 (np.ndarray): The image for frame t (previous), with lines and labels drawn.
2. img2 (np.ndarray): The image for frame t+1 (current), with matching lines and labels.
3. frame_idx (int): The current frame index, shown at the bottom of the figure for reference.

Output:

None (displays the image pair using matplotlib.pyplot).

collate_fn_variable_lines(batch)

Prepares a batch of images and variable-length line data for a DataLoader.Used in PyTorch DataLoader when lane line counts vary across images.

Input:

batch: A list of (image_tensor, lines_tensor) tuples

Output:

images_tensor: A stacked tensor of shape [B, 3, H, W]
lines_list: A list of tensors, each of shape [N, 4], for lane lines

visualize_prediction(lines_Dataloader, association_model=None)

Visualizes the matching of lane lines across consecutive video frames using either a rule-based method or a neural network model. Each match is color-coded and annotated with a unique ID for tracking.

Inputs:

1. lines_Dataloader: A PyTorch DataLoader that yields one image and its lane lines per iteration.
2. Each batch contains: image_tensor of shape [3, H, W],lines_tensor of shape [N, 4]
3. association_model (optional): A trained model used to predict matching lines. If None, a rule-based method (e.g. cosine similarity, distance threshold) is used instead.

Outputs: No return value.
The function displays:

1. Two annotated frames (t and t+1) side by side
2. Color-coded line matches
3. Printed match IDs in the console

Use Cases:

1. Debugging line association models
2. Visual validation of matching quality
3. Demo or presentation of tracking results

evaluate_association_methods(lines_Dataloader, association_model, num_batches=10)
Evaluates and compares the performance of a rule-based line matching method (association_lines) and a neural network-based method (predict_matches_nn) over a fixed number of frame pairs.

Inputs:
1. lines_Dataloader: A PyTorch DataLoader that yields (image, lines) tuples for each frame.
2. association_model: A trained neural network model that predicts line matches.
3. num_batches (int, optional): Number of frame pairs to process for evaluation (default: 10).

Outputs: No return value.

Main Execution Block **(if __name__ == "__main__":)**
This block defines the top-level logic of the script and controls the workflow based on the user's selected mode (training, visualization, or evaluation). It acts as the main entry point when the script is run directly.

Inputs (via flags):
1. train (bool): If True, extracts features, trains the neural line association model, and saves it.
2. visualize (bool): If True, visualizes line matching between frames.Uses the neural model if provided.Falls back to rule-based matching if association_model is None
3. evaluate (bool):If True, compares neural model predictions with rule-based matching.
Requires a pre-trained model.

Data Handling:
Loads dataset splits (train.txt, val.txt, test.txt) from the CULane dataset using a helper function get_loader(...).
Applies custom collate_fn if needed for variable-length line inputs.

Outputs:

1. If train = True: Saves the trained model to disk (association_model_final.pth)
2. If visualize = True: Displays matched lane lines between consecutive frames using matplotlib
3. If evaluate = True: Prints accuracy (agreement) and runtime statistics comparing neural vs. rule-based matches

Notes:
1. visualize can run without a trained model using rule-based matching
2. evaluate requires the model to be trained beforehand
3. The flags are mutually independent , you can turn on one or multiple depending on your goal