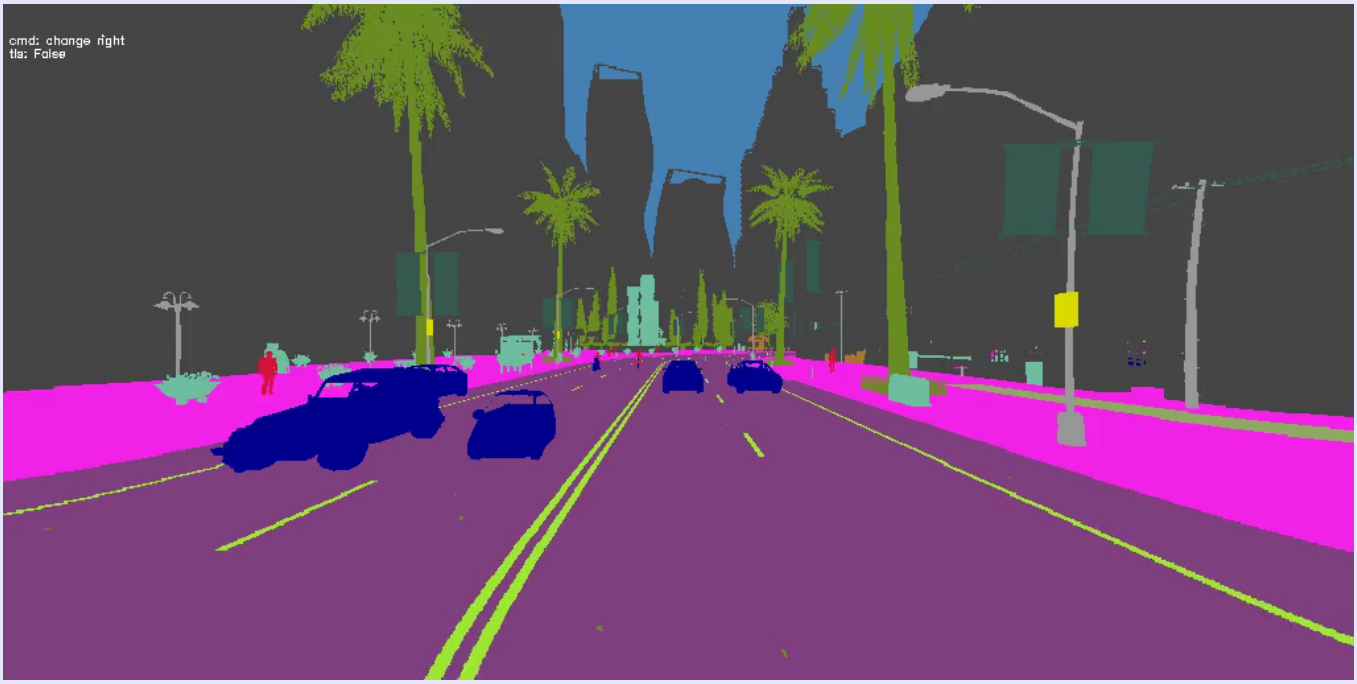
Unsupervised Semantic Segmentation for Autonomous Driving Cars



Felix L. Hoekstra

Layout: typeset by the author using LaTeX.
Cover illustration: Carla Simulator

Unsupervised Semantic Segmentation for Autonomous Driving Cars

Felix L. Hoekstra
11330600

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 900
1098 XH Amsterdam

Supervisor
Dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

Semester 2, 2023-2024

Abstract

This thesis investigates the application of unsupervised semantic segmentation for autonomous driving cars, focusing on the Self-Supervised Transformer with Energy-based Graph Optimization (STEGO) algorithm. Semantic segmentation, crucial for autonomous vehicles, assigns class labels to each pixel, enabling detailed environmental understanding. The study employs the DINO model for feature extraction, followed by feature correspondence computation, clustering, and refinement using Conditional Random Fields (CRFs). Various methodologies, including frame-by-frame and batch processing on both GPU and CPU, were explored to achieve real-time segmentation. Despite significant performance improvements through mixed precision and optimized memory management, achieving fully real-time segmentation remains a challenge. The thesis concludes with reflections on the current limitations and proposes future directions, including advanced initialization techniques, refined loss functions, and enhanced data augmentation, to improve segmentation quality and real-time processing capabilities.

Acknowledgement

I thank Arnoud Visser of the *Institute of Informatics* at the Universiteit van Amsterdam for providing the opportunity to work on this project and guiding me along the way. His advice and patience are most appreciated, as well as the provision of the RoboLab as a workplace.

Contents

Chapter 1

Introduction

The development of autonomous vehicles has rapidly progressed, with significant advancements in sensor technology, data processing capabilities, and machine learning algorithms. A critical component of autonomous driving systems is the ability to accurately interpret and understand the surrounding environment through semantic segmentation. This process involves classifying each pixel in an image captured by vehicle-mounted cameras into predefined categories such as roads, vehicles, pedestrians, and other relevant objects.

The progress in sensor technology, such as high-definition cameras, LiDAR, and radar systems, has greatly enhanced the perception capabilities of autonomous vehicles. These sensors provide a comprehensive understanding of the vehicle's surroundings, enabling the detection and classification of various objects in real-time [16, 7]. The fusion of data from multiple sensors further improves the accuracy and reliability of the perception system, allowing for more robust and efficient autonomous driving [16].

One approach to semantic segmentation in autonomous vehicles is Bird's Eye View (BEV) semantic segmentation. BEV refers to a top-down view of the scene, which provides a comprehensive perspective of the vehicle's surroundings. This technique transforms the captured images into a bird's eye view, making it easier to detect and classify objects accurately. BEV semantic segmentation is particularly useful for understanding the spatial relationships between objects and navigating complex environments, offering enhanced spatial context and facilitating better decision-making for autonomous systems [9, 14].

Machine learning algorithms, particularly deep learning techniques, have also played a crucial role in advancing autonomous vehicle technology. These algorithms are capable of processing vast amounts of data to learn complex patterns and make accurate predictions about the vehicle's environment. Semantic segmentation, powered by these advanced algorithms, is essential for identifying and

understanding different elements in the driving scene, which is crucial for safe and efficient navigation [16, 7].

This thesis focuses on implementing a segment neural network for self-driving cars, leveraging the capabilities of the CARLA simulator and state-of-the-art segmentation algorithms. The primary goal is to enhance the accuracy and reliability of semantic segmentation in dynamic driving environments, such that in the future video feed can be segmented in real-time will perform better than the current alternatives.

Chapter 2

Related Work

2.1 Competitors

2.1.1 Learning by Cheating (LBC)

Learning by Cheating (LBC) is a method that utilizes a privileged agent (teacher) with access to BEV semantic segmentation images to train a sensorimotor agent (student) using standard RGB images. This approach separates the learning process into learning to act and learning to perceive, leveraging privileged information during training to simplify and improve the learning process.

Methodology

LBC employs a teacher-student architecture:

- **Privileged Agent (Teacher):** Has access to BEV semantic segmentation images and learns to act based on this privileged information.
- **Sensorimotor Agent (Student):** Trained to imitate the teacher's actions using only forward-facing RGB images, without needing privileged information during deployment.

The teacher agent learns from a dataset of expert trajectories, and the student agent is trained under the supervision of the teacher, learning an end-to-end policy from the privileged actions.

Performance

LBC significantly reduced traffic light infractions and collisions compared to state-of-the-art methods, demonstrating superior performance in autonomous driving

tasks. This approach provided a robust framework for training autonomous driving agents using privileged information.

2.1.2 World on Rails (WoR)

World on Rails (WoR) is a model-based reinforcement learning method developed for autonomous driving. WoR simplifies the autonomous driving problem by assuming a static environment, known as the "world-on-rails" assumption. This method decouples the agent's actions from the world's reactions, allowing for a more straightforward learning process.

Methodology

WoR employs a model-based approach where:

- **Ego-Vehicle Model:** A model of the vehicle is trained to predict its next state based on its current state and action, enabling the simulation of the vehicle's behavior without actual environmental interaction.
- **World Model:** The environment's states are pre-determined, independent of the agent's actions. This model generates a sequence of world states that are not influenced by the vehicle's actions.

The Q-values, representing the expected reward of actions, are computed using a predefined reward function. This function rewards lane-keeping and penalizes collisions and traffic infractions. Dynamic programming and backward induction are used to compute Q-values for all possible states and actions, which are then used to train the visuomotor policy.

Performance

WoR demonstrated significant improvements in autonomous driving benchmarks, achieving a 25% higher driving score on the CARLA leaderboard using 40 times less data compared to other methods. It showed excellent performance on the NoCrash benchmark and generalized well across different environments using the ProcGen platform.

2.1.3 Cheating by Segmentation (CBS)

Cheating by Segmentation (CBS) is an approach that adapts the Learning by Cheating (LBC) method for more practical applications by using forward-facing camera images instead of bird's-eye view (BEV) images. This adaptation aims

to make the training process more applicable to real-world scenarios by utilizing readily available sensory inputs.

Methodology

CBS employs a teacher-student architecture:

- **Teacher:** A privileged agent with access to ground truth BEV semantic segmentation images, which predicts waypoints based on this data.
- **Student:** An agent trained to use forward-facing RGB images to mimic the teacher's predictions, enabling the student to operate without privileged BEV information.

The teacher agent learns optimal actions from expert trajectories, while the student agent learns to predict these actions using only RGB images. This method leverages the practical availability of forward-facing cameras to train robust autonomous driving agents.

Performance

CBS demonstrated improved practicality by utilizing forward-facing cameras while maintaining the core advantage of using privileged information for training. This approach effectively bridged the gap between theoretical models and real-world applications.

2.1.4 Cheating by Segmentation 2 (CBS2)

CBS2 is an extension of the original Cheating by Segmentation (CBS) approach, designed to train autonomous driving agents using forward-facing cameras instead of bird's-eye view (BEV) images. The primary aim is to leverage large existing datasets like Waymo without needing costly and hard-to-obtain BEV images.

Methodology:

CBS2 employs a teacher-student architecture:

- **Teacher:** A privileged agent with access to ground truth BEV semantic segmentation images, which predicts waypoints based on this data.
- **Student:** An agent trained to use forward-facing RGB images to mimic the teacher's predictions, enabling the student to operate without privileged BEV information.

The training involves the teacher learning optimal actions from expert trajectories and the student learning to predict these actions using only RGB images. CBS2 was evaluated using the CARLA simulator, with different student model versions (e.g., Pyramid Pooling Modules (PPM) or Feature Pyramid Networks (FPN)) tested to improve perception and reduce infractions.

Performance:

CBS2 outperformed previous methods like Learning by Cheating (LBC) and World on Rails (WoR) in lane-keeping and collision rates. However, it still faced challenges in accurately predicting braking actions, leading to more collisions and traffic light infractions than LBC on the NoCrash benchmark.

2.1.5 Contribution to This Work

In this thesis, CBS2, WoR, LBC, and CBS provided foundational methodologies for data collection and model training. CBS2's approach for optimizing data collection parameters, such as the number of frames, steering noise, and traffic light detection distance, was instrumental in creating a realistic and challenging dataset. WoR's static environment assumption simplified the learning process, allowing for efficient policy training using pre-recorded driving logs. LBC and CBS's teacher-student architecture informed the design of this segmentation framework, leveraging privileged information to enhance the accuracy and reliability of semantic segmentation in dynamic driving environments. For a clear visualization and overview of all the related work and their relation to this work see appendix A.3.

2.2 Related Theory

2.2.1 Sensor Fusion and Data Collection

The effectiveness of semantic segmentation in autonomous vehicles is significantly enhanced by sensor fusion, which combines data from multiple sensors to provide a comprehensive understanding of the environment. Sensors such as cameras, LiDAR, and radar each have unique strengths and limitations. By integrating data from these sensors, the system can achieve a more robust and accurate perception of its surroundings [2, 15].

In the context of this project, the CARLA simulator is used for data collection. CARLA provides a realistic simulation environment that replicates urban driving scenarios, allowing for the collection of high-quality, labeled data necessary for

training segmentation models. The simulator supports various sensor configurations, enabling the generation of diverse datasets that reflect real-world conditions [2].

2.2.2 Model Training and Evaluation

Training semantic segmentation models involves using large datasets to learn the mapping from input images to pixel-wise class labels. The training process typically includes data augmentation techniques to enhance the model's robustness and generalization capabilities. Evaluation metrics such as Intersection over Union (IoU) and mean IoU are commonly used to assess the performance of segmentation models [2, 15].

2.2.3 Dino

DINO stands for Self-Distillation with No Labels. It is a self-supervised learning approach that leverages Vision Transformers (ViT) to generate feature representations without the need for labeled data. The key aspects of DINO that make it suitable for this task are [1]:

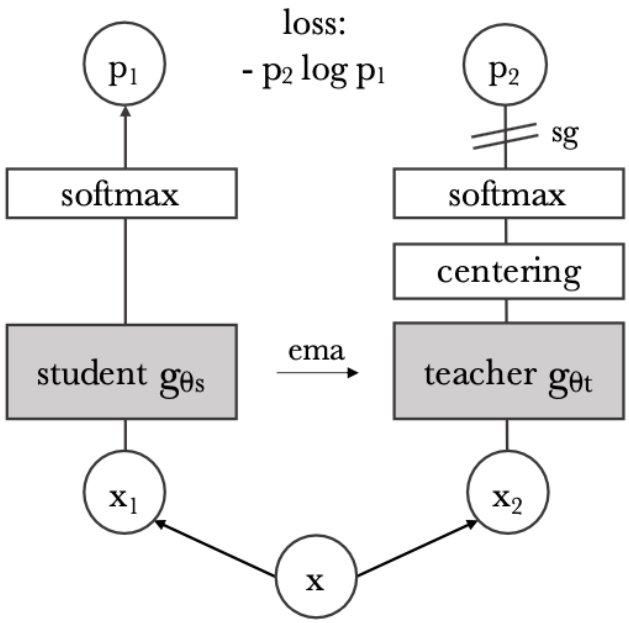


Figure 2.1: DINO Overview

Self-supervised Learning

DINO uses self-supervised learning techniques to train Vision Transformers. Self-supervised learning allows the model to learn from the inherent structure of the data itself, without the need for manual annotations. This makes DINO highly versatile and capable of generalizing across different types of visual data.

Vision Transformers (ViT)

Vision Transformers apply the transformer architecture, which is known for its ability to capture long-range dependencies, to images. This makes them particularly effective for tasks like semantic segmentation, where understanding the context and global information within the image is important.

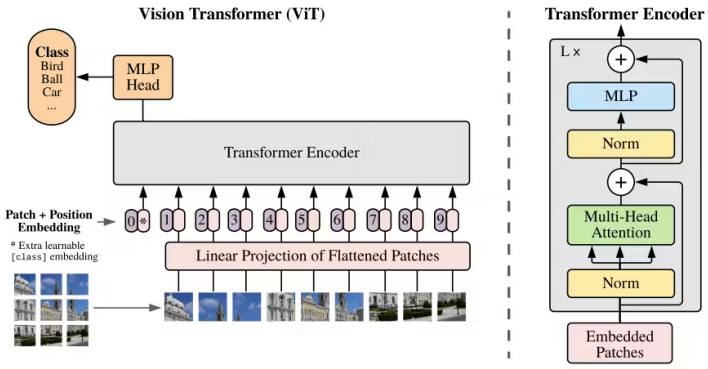


Figure 2.2: ViT transformer example

High-quality Feature Representations

DINO has been shown to produce high-quality feature representations that are semantically meaningful. These features are crucial for the STEGO framework, as they provide a solid foundation for the clustering and segmentation tasks that follow.

DINO has demonstrated state-of-the-art performance in various vision tasks, making it a reliable choice for the feature extraction step in the STEGO framework.[12]

2.2.4 CRFs

Conditional Random Fields (CRFs) are a type of probabilistic graphical model used to model structured relationships in data. Unlike simpler models that treat each

data point independently, CRFs consider the relationships between neighboring data points, making them powerful for tasks where context and dependencies are important, such as image segmentation or natural language processing.

A CRF is an undirected graph where each node represents a random variable, and edges represent dependencies between these variables. In the context of image segmentation, each node could represent the label of a pixel, and edges represent the relationships between neighboring pixels.

The goal of a CRF is to predict a label for each node (pixel) by considering both the local information (e.g., the intensity of a pixel) and the contextual information from neighboring nodes (pixels). See figure 2.3

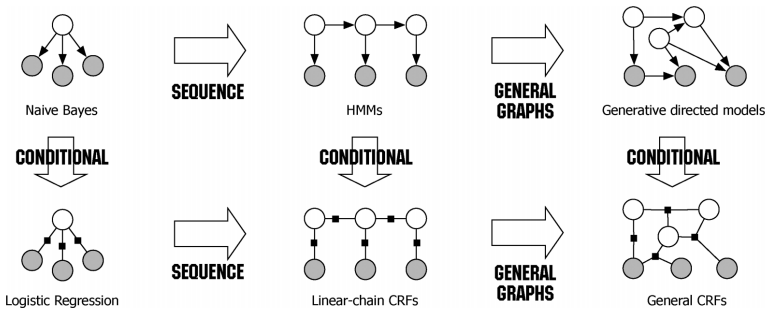


Figure 2.3: CRF Visualization

Chapter 3

Theoretic Approach

3.1 Semantic Segmentation in Autonomous Vehicles

Semantic segmentation is a pivotal component in the field of computer vision, particularly for autonomous driving systems. It involves assigning a class label to each pixel in an image, thus enabling the vehicle to perceive and understand its environment. This granular level of understanding is essential for the vehicle to navigate safely and efficiently through complex and dynamic environments.

The theoretical foundation of semantic segmentation lies in deep learning, specifically convolutional neural networks (CNNs). CNNs are highly effective at capturing spatial hierarchies in images, making them suitable for tasks that require detailed spatial analysis, such as segmentation [8, 11]. State-of-the-art models for semantic segmentation include Fully Convolutional Networks (FCNs), U-Net, and DeepLab, each designed to improve the accuracy and efficiency of segmentation tasks by utilizing various architectural innovations [8, 5].

3.2 Self-Supervised Transformer with Energy-based Graph Optimization

Self-Supervised **T**ransformer with **E**nergy-based **G**raph **O**ptimization (STEGO), introduced in 2022, aims to discover and localize semantically meaningful categories within image corpora without any form of annotation.[6] STEGO is particularly well-suited for real-time segmentation of video data from simulators like Carla, thanks to its unsupervised learning capabilities. Below are the core steps of the algorithm denoted.

1. **Feature Extraction with Frozen Backbone**

The first step in the STEGO (Self-supervised Transformer with Energy-based Graph Optimization) algorithm involves extracting dense features from input images using a pre-trained unsupervised model. The choice of the model is critical, as the quality and relevance of the extracted features directly influence the subsequent steps of the algorithm.

STEGO utilizes the DINO (Self-Distillation with No Labels) model, a visual transformer known for its ability to produce high-quality and semantically consistent features. These features are extracted from intermediate layers of the model, which capture detailed and spatially dense information about the input images. The use of dense feature maps allows for a granular representation of image content, which is essential for effective semantic segmentation. See the global overview of this step below in figure 3.1

Figure 3.1: Feature Extraction with Frozen Backbone

Detailed Process

(a) **Pre-trained Model Selection**: The algorithm employs a pre-trained DINO model, chosen for its state-of-the-art performance in self-supervised learning tasks. DINO's ability to generate semantically meaningful features without labels makes it an ideal candidate for unsupervised segmentation [6].

(b) **Feature Extraction**: Input images x are fed into the pre-trained DINO model to obtain dense feature maps. Let f represent the feature tensor extracted from an input image. The feature tensor f has dimensions $\mathbb{R}^{C\times H\times W}$, where C is the number of feature channels, and H and W are the height and width of the feature map, respectively.

$$f = N(x) \tag{3.1}$$

Here, $N : \mathbb{R}^{C'\times H'\times W'} \to \mathbb{R}^{C\times H\times W}$ is the mapping function of the frozen backbone. C', H', and W' are the dimensions of the input image, and C, H, and W are the dimensions of the output feature tensor [6].

(c) **Freezing the Backbone**: The weights of the pre-trained DINO model are kept frozen during the entire training process of STEGO. This means that the feature extraction model is not fine-tuned or updated; it remains static. Freezing the backbone ensures that the feature representations are consistent and stable, allowing the segmentation head to focus on learning how to cluster and refine these features [**?**].

(d) **Semantic Consistency**: The extracted features are noted for their semantic consistency. This means that similar features across different images or different regions within the same image represent similar semantic content, a property that is crucial for the success of the STEGO algorithm [6].

2. **Compute Feature Correspondence**

After extracting dense features using a pre-trained model like DINO, the next crucial step in the STEGO algorithm is to compute feature correspondences. This step forms the foundation for understanding the relationships between different parts of the image or between different images, which is essential for semantic clustering.

The computation of feature correspondences involves calculating the cosine similarity between features of different images or different regions within the same image. This similarity is captured in the form of a feature correspondence tensor. For clarification see figure 3.2

Detailed Process

(a) **Feature Correspondence Calculation**: For each pair of feature tensors f and g extracted from two different images (or different regions within the same image), the cosine similarity between the features is computed. This is formalized as the feature correspondence tensor F.

$$F_{hwij} := \sum_{c} \frac{f_{chw}}{|f_{hw}|} \frac{g_{cij}}{|g_{ij}|} \tag{3.2}$$

Here, f and g represent the feature tensors for two different images, where C is the number of channels, and H, W, I, J are the spatial dimensions. The entries of F represent the cosine similarity between the feature at spatial position (h, w) of tensor f and position (i, j) of tensor g[6].

(b) **Utilizing Correlation Volumes**: The correlation volume approach, especially for convolutional or transformer architectures, is particularly

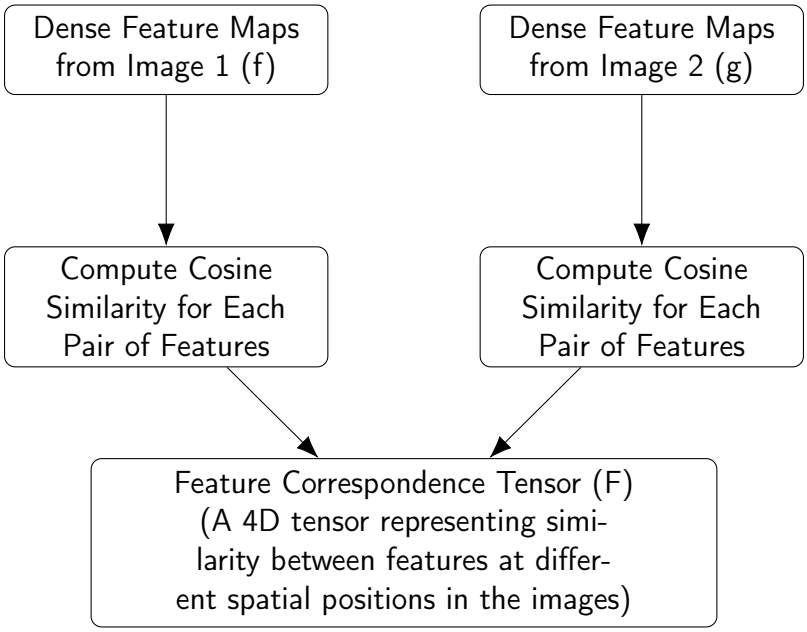


Figure 3.2: Compute Feature Correspondence

useful. Dense feature maps from specific layers are used as the activation map. Although Q, K, or V matrices in transformers can also serve as candidate features, they are found to be less effective in practice.

(c) **Special Case of Self-Correspondence**: When $f = g$, the feature correspondence tensor measures the similarity between two regions within the same image. This self-correspondence is crucial for understanding intra-image feature relationships, which can be used to enhance segmentation accuracy.

(d) **Visualization and Interpretation**: By examining slices of the correspondence tensor F, it is possible to visualize how different images or regions relate according to the feature extractor. This visualization helps in understanding the semantic consistency of features across the image corpus. This method acts as a higher-order generalization of Class Activation Maps for contrastive architectures and visual search engines.

3. **Distill Correspondences**

Once feature correspondences are computed, the next step in the STEGO algorithm is to distill these correspondences into discrete semantic labels. This process involves refining the raw correspondences into a more structured form that can be effectively used for segmentation.

Distilling correspondences is achieved using a contrastive loss function, which encourages the formation of compact clusters in the feature space while preserving the semantic relationships between features. Graph optimization techniques are integral to this process, ensuring that the feature correspondences are refined into meaningful and discrete clusters. See the details below in figure 3.3

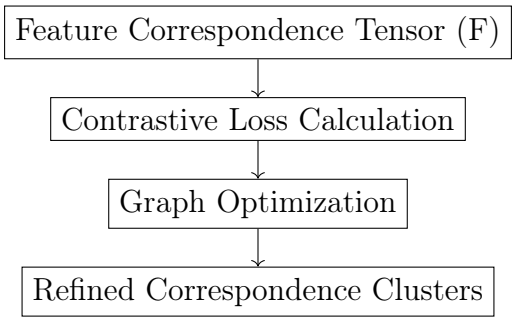


Figure 3.3: Distill Correspondences

Detailed Process

(a) **Segmentation Feature Mapping**: For each pair of feature tensors f and g from two images x and y, respectively, these features are mapped into a lower-dimensional space using a segmentation head S. The resulting segmentation features are $s := S(f) \in \mathbb{R}^{CHW}$ and $t := S(g) \in \mathbb{R}^{CIJ}$.

(b) **Feature Correlation Tensor**: Using the previously computed feature correspondence tensor F and the newly obtained segmentation features s and t, a segmentation correlation tensor S is constructed. This tensor is computed by element-wise multiplication of F and the segmentation features.

(c) **Contrastive Loss Function**: The core of this distillation process is a contrastive loss function designed to push similar features together and dissimilar features apart. The loss function is defined as:

$$L_{simple-corr}(x, y, b) := -\sum_{hwij}(F_{hwij} - b)S_{hwij} \tag{3.3}$$

Here, b is a hyper-parameter that adds uniform negative pressure to the equation to prevent collapse. This loss function encourages the elements of S to be large when elements of $F - b$ are positive and small when they are negative [**?**].

(d) **Graph Optimization**: The loss function incorporates graph optimization techniques to ensure that the feature correspondences are refined into discrete clusters. This involves optimizing the segmentation features to form compact clusters, leveraging the structure of the feature correlation tensor.

(e) **Optimization and Stability**: To ensure stable optimization, weakly-correlated segmentation features are optimized to be orthogonal, which can be efficiently achieved by clamping the segmentation correspondence S at 0. This step dramatically improves optimization stability.

(f) **Spatial Centering**: To balance the learning signal for small objects with concentrated correlation patterns, a spatial centering operation is applied on the feature correspondences:

$$FSC_{hwij} := F_{hwij} - \frac{1}{IJ} \sum_{i'j'} F_{hwi'j'} \tag{3.4}$$

This modification ensures a more balanced optimization process, leading to better segmentation results.

(g) **Final Correlation Loss**: The final correlation loss function, incorporating the above modifications, is defined as:

$$L_{corr}(x, y, b) := -\sum_{hwij} (FSC_{hwij} - b) \max(S_{hwij}, 0) \tag{3.5}$$

4. **Train Segmentation Head**

Following the distillation of feature correspondences, the next step in the STEGO algorithm is to train the segmentation head. The purpose of this step is to project the distilled feature correspondences into a lower-dimensional embedding space, where they can form distinct clusters that correspond to different semantic regions in the image. Training the segmentation head is crucial as it refines the feature relationships and ensures that the clusters formed are both compact and semantically meaningful. The segmentation head is trained using a combination of correspondence losses to optimize its performance. See figure 3.4a

The segmentation head is a lightweight feed-forward network with ReLU activations. This network maps the high-dimensional feature space into a lower-dimensional code space where $K < C$, with K being the dimensions of the code space and C being the number of channels in the feature tensor. The training process involves using three types of correspondence losses: self-correspondence loss, K-Nearest Neighbors (KNN) correspondence loss, and

random image correspondence loss. The self-correspondence loss measures the correspondence between an image and itself, providing a positive, attractive signal. The KNN correspondence loss measures the correspondence between an image and its KNNs, primarily providing a positive, attractive signal. The random image correspondence loss measures the correspondence between an image and random other images, providing a negative, repulsive signal.

The combined loss function for training the segmentation head is given by:

$$L = \lambda_{\text{self}} L_{\text{corr}}(x, x, b_{\text{self}}) + \lambda_{\text{knn}} L_{\text{corr}}(x, x_{\text{knn}}, b_{\text{knn}}) + \lambda_{\text{rand}} L_{\text{corr}}(x, x_{\text{rand}}, b_{\text{rand}}) \tag{3.6}$$

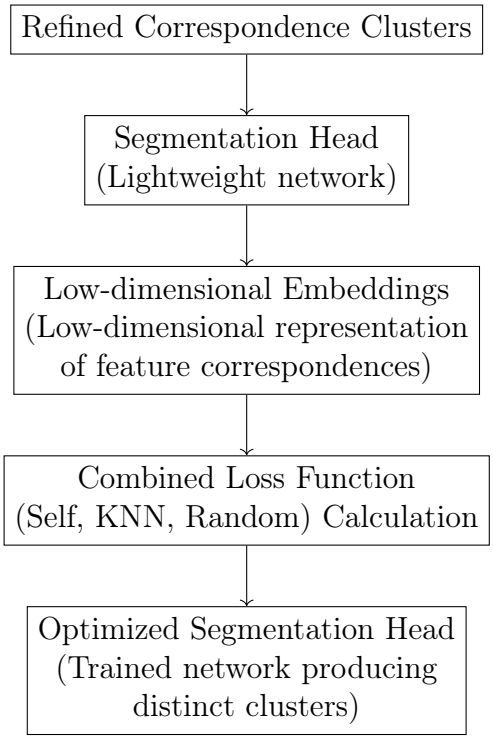
Here, λ terms control the balance of the learning signals, and b terms control the ratio of positive to negative pressure [**?**]. The optimization process involves adjusting the parameters of the segmentation head to minimize the combined loss function. This ensures that the segmentation features form compact and semantically meaningful clusters.

Each training minibatch consists of a collection of random images, their KNNs, and random other images. Five-crop augmentation is applied to enhance the training process, allowing the network to look at closer details of the images. This multi-faceted approach to loss calculation and the use of a lightweight segmentation head ensures that the segmentation model can accurately project the distilled feature correspondences into distinct, meaningful clusters that are critical for effective semantic segmentation.

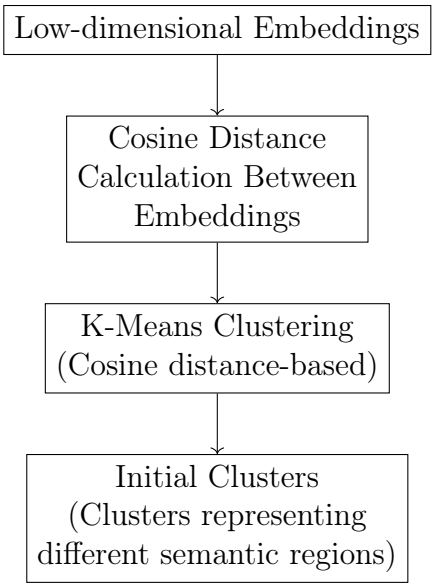
5. **Cluster Formation**

Following the training of the segmentation head, the next step in the STEGO algorithm is cluster formation. This step is critical as it involves using the low-dimensional embeddings produced by the segmentation head to create distinct semantic clusters. The formation of clusters from these embeddings is essential for translating the continuous feature space into discrete semantic labels that can be used for segmentation.

The clustering process leverages a cosine distance-based K-Means algorithm to form the initial clusters. See figure 3.4b. The choice of K-Means is motivated by its effectiveness in handling high-dimensional data and its ability to form compact and well-separated clusters. The algorithm works by iteratively assigning data points to the nearest cluster centroid and then updating the centroids based on the mean of the assigned points. This process continues until the cluster assignments no longer change significantly.



(a) Training the Segmentation Head



(b) Cluster Formation Process in STEGO

Figure 3.4: (a) Cluster Formation Process in STEGO (b) Training the Segmentation Head

The specific steps involved in the clustering process are as follows:

First, the low-dimensional embeddings $s \in \mathbb{R}^{K \times H \times W}$ produced by the segmentation head are collected. These embeddings represent the distilled feature correspondences in a lower-dimensional space. The cosine distance between the embeddings is then computed to determine their similarity. The cosine distance is chosen because it measures the cosine of the angle between two non-zero vectors, providing a measure of orientation rather than magnitude.

Next, a minibatch K-Means algorithm is applied to the embeddings. The algorithm initializes with a set of random cluster centroids. Each embedding is then assigned to the nearest centroid based on the cosine distance. The centroids are updated by calculating the mean of the embeddings assigned to each cluster. This assignment and update process is repeated iteratively until the cluster assignments stabilize.

The clustering process results in a set of discrete clusters, each representing

a different semantic region in the image. These clusters form the basis for the initial segmentation labels. The use of K-Means ensures that the clusters are compact and well-separated, providing a robust foundation for the subsequent refinement step.

6. **Refinement**

   After the initial clusters are formed using the K-Means algorithm, the next step in the STEGO algorithm is to refine these clusters to improve their spatial coherence and accuracy. This refinement step is critical for ensuring that the segmentation results are smooth and adhere closely to the boundaries of objects within the image. The refinement process leverages Conditional Random Fields (CRF) to achieve this goal.

   First, the initial clusters obtained from the K-Means algorithm are used as input to the CRF, as can be seen in figure 3.5. The CRF model considers the pairwise relationships between neighboring pixels to refine the cluster assignments. This is achieved by optimizing an energy function that combines both unary and pairwise potentials. The unary potential represents the likelihood of a pixel belonging to a particular cluster based on the initial clustering results. The pairwise potential, on the other hand, encodes the relationship between neighboring pixels, encouraging pixels with similar features to be assigned the same cluster.

   The energy function for the CRF is defined as follows:

$$E(x) = \sum_i \psi_u(x_i) + \sum_{i,j} \psi_p(x_i, x_j) \tag{3.7}$$

   Here, $\psi_u(x_i)$ is the unary potential, and $\psi_p(x_i, x_j)$ is the pairwise potential. The unary potential is derived from the initial cluster assignments, while the pairwise potential is typically defined based on the spatial and color similarity between neighboring pixels. The optimization of this energy function results in refined cluster assignments that are spatially coherent.

   To further enhance the refinement process, the CRF model may be augmented with additional features such as color, texture, and edge information. This allows the model to better capture the boundaries of objects within the image, leading to more accurate segmentation results. The refinement process iterates over the entire image, updating the cluster assignments until convergence is achieved.

7. **Output Segmentation**

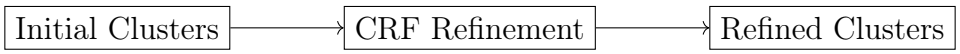


Figure 3.5: Refinement Process

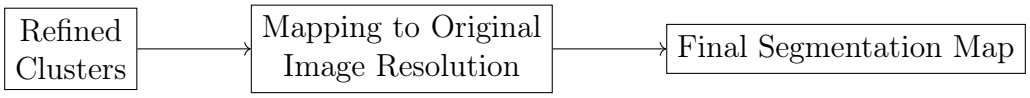


Figure 3.6: Output Segmentation Process

The final step in the STEGO algorithm is to produce the output segmentation maps. After refining the clusters using Conditional Random Fields (CRF), the algorithm maps these refined clusters back to the original image space, resulting in detailed and accurate segmentation maps. For the specific step see figure 3.6 and for the complete overview see figure 3.7

The output segmentation step is crucial as it translates the refined feature clusters into concrete class assignments for each pixel in the image. This process involves mapping the low-dimensional embeddings and refined clusters back to the original high-resolution image space, ensuring that the final segmentation adheres closely to the object boundaries and semantic regions within the image.

First, the refined clusters obtained from the CRF process are used to generate the final segmentation labels. Each pixel in the image is assigned to a specific cluster based on the refined feature correspondences. This assignment ensures that pixels with similar features and spatial relationships are grouped together into meaningful semantic categories.

Next, the algorithm projects these cluster assignments back to the original image resolution. This step is essential for maintaining the high spatial resolution and detail required for accurate segmentation. The projection involves mapping the low-dimensional embeddings and cluster assignments from the feature space to the pixel space of the original image.

The final segmentation map is produced by assigning each pixel to the semantic category represented by its corresponding cluster. This mapping ensures that the segmentation labels are consistent with the refined clusters and adhere to the boundaries of objects within the image. The output segmentation map is a high-resolution representation of the semantic regions within the image, providing a detailed and accurate segmentation of the visual content.

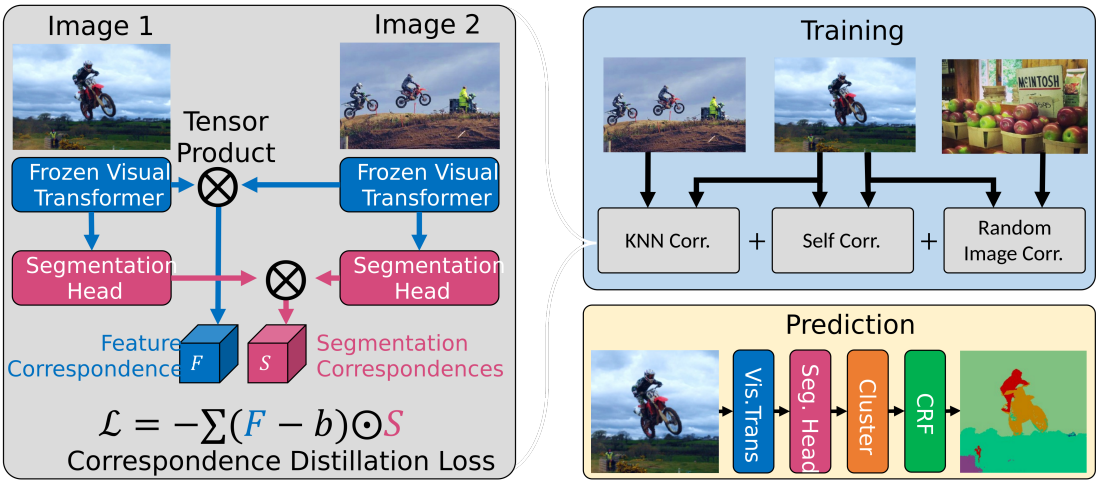


Figure 3.7: Stego Architecture Overview

Chapter 4

Methodology

In this thesis, I aim to enhance the accuracy and efficiency of real-time semantic segmentation using the CARLA simulator for autonomous driving applications. The primary focus is on improving the data collection process, refining segmentation algorithms, and evaluating their performance under various conditions. WoR [3], LBC [4], and CBS [10] laid the foundation where CBS2 [13] fine-tuned the CARLA environment and data collection phase. CBS2 improved upon CBS by optimizing the parameters used for data collection, such as the number of frames, the steering noise, and the traffic light detection distance. These enhancements allowed for a more realistic and challenging dataset, crucial for training robust segmentation models. This approach leverages these advancements to implement a segmentation network that can perform efficiently in diverse driving scenarios, with an emphasis on real-time processing and accuracy in varied environmental conditions.

The STEGO network is trained using a dataset collected in a simulated environment that closely mimics real-world driving conditions. The dataset includes a variety of weather conditions, traffic densities, and pedestrian interactions, ensuring comprehensive coverage of potential real-world scenarios and objects. This training process is designed to refine the network's ability to discern and segment different objects and road features accurately, enhancing the overall performance of autonomous driving systems.

4.1 Carla Simulator

The Carla Simulator provides various towns and maps, each with a distinct layout, possible scenarios, and routes. Additionally, a wide range of weather and traffic conditions can be simulated, ensuring a driving environment as realistic as possible.

4.2 Data Collection

In the context of unsupervised learning, a critical challenge is the evaluation of the eventual output. Fortunately, the Carla Simulator is equipped with multiple sensors, one of which is a semantic segmentation feed. This feed can provide the necessary ground truth data for comparing the model's results, enabling a comprehensive evaluation. The data collection was executed in two phases: training and testing, each with a distinct approach.

4.2.1 Phase 1: Training Data Collection

In the context of unsupervised learning, a critical challenge is the evaluation of the eventual output. Fortunately, the Carla Simulator is equipped with multiple sensors, one of which is a semantic segmentation feed. This feed can provide the necessary ground truth data for comparing the model's results, enabling a comprehensive evaluation. The data collection was executed in two phases: training and testing, each with a distinct approach. Evaluating driving episodes based on resolution and accuracy was challenging due to the lack of direct feedback during data collection. To address this, a logging function was implemented to record each episode of a data collection run. This approach saved significant time, as it was later discovered that CBS2 had left augmentation settings in the video feed, resulting in faulty data. For details on these augmentations and image errors, please refer to Appendix A.6.

Scenarios & Routes

The approach for the training phase was to use a town that could provide a variety of environmental factors to enrich the collected data with the noted settings. After some consideration, Town 04 of Carla showed considerable promise as it boasts highway, urban, and outdoor scenes, in addition to ample scenarios and routes provided by CBS2 (see Figure 4.1). For visualization there is a example route provided in figure 4.2. Due to the fact that these waypoints had to be drawn with manual offset they do not perfectly match the roads in the shown map.

Data Preparation

The collected data was stored in LMDB format, which is optimized for efficiency but not directly usable by the STEGO model. To prepare the data, it was necessary to read and convert it into a specific format. The LMDB contained both RGB and semantic segmentation channels. To ensure the process was both fast and memory-efficient, a function was implemented using CBS2-related data loader classes. This

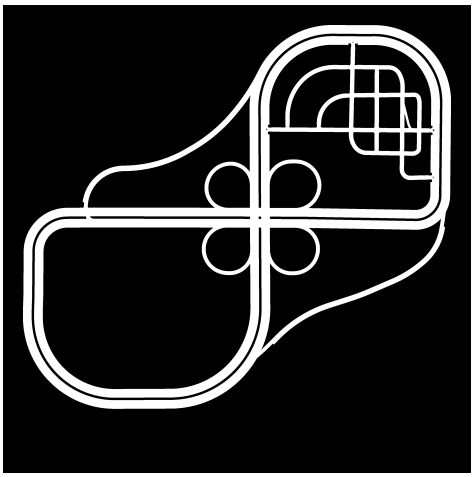


Figure 4.1: Map of Town 04 in CARLA.

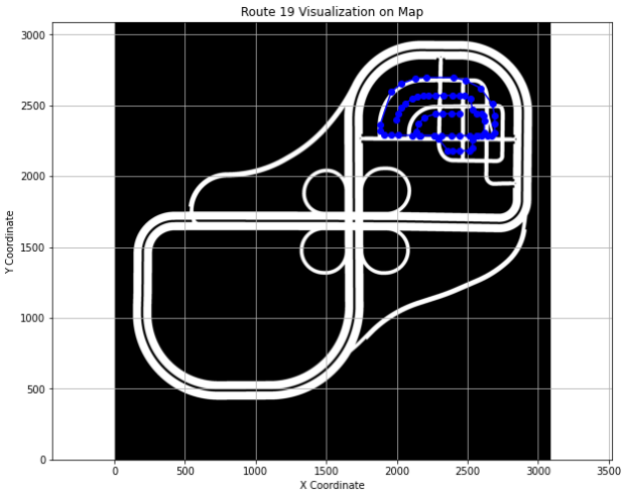


Figure 4.2: Visualization of a route for Town 04

function extracted each frame from a driving episode, saving RGB frames as JPEGs and segmentation frames as PNGs, as required by STEGO, see figure 4.3. It is important to note that the labels here are only needed for evaluation purposes, not for training, due to the unsupervised nature of the model.

To streamline the image processing workflow, parallel processing jobs are initiated based on the provided dataset size. For larger datasets, parallel processing is employed to speed up the execution by utilizing multiple CPU cores. Additionally, garbage collection is triggered periodically to clear unused memory, preventing memory leaks and optimizing resource usage. However, for smaller datasets, parallel processing is avoided. This is because the overhead of initializing and managing parallel tasks can outweigh the benefits when the dataset is small. The overhead includes the time required to start multiple processes, distribute tasks, and gather results, which can be significant compared to the time spent on actual processing. As a result, processing small datasets sequentially is more efficient, reducing the complexity and resource contention that might occur with parallel processing. Furthermore, error handling mechanisms are implemented to ensure robust processing, particularly for test datasets where parallel processing might occasionally fail. In such cases, the script falls back to sequential processing, ensuring that all images are processed without interruption

4.2.2 Phase 2: Testing Data Collection

Next to the first data collection with a semi-high resolution, to ensure the training phase would be accurate enough while still being reasonably fast, there was also a

```
dataset_name
|— imgs
|   ├— train
|   |   |— unique_img_name_1.jpg
|   |   └— unique_img_name_2.jpg
|   └— val
|       |— unique_img_name_3.jpg
|       └— unique_img_name_4.jpg
└— labels
    ├— train
    |   |— unique_img_name_1.png
    |   └— unique_img_name_2.png
    └— val
        |— unique_img_name_3.png
        └— unique_img_name_4.png
```

Figure 4.3: STEGO directory format for dataset

second data collection that had higher resolution to use for the test segmentation. Whereas, the resolution for the testing phase was set to 1120 and 560, respectively. The decision for these values is critical as it represents a multiple of the patch size (and cropping resolution) used in the STEGO model. See the STEGO methodology for elaboration.

Scenarios & Routes

Where town 4 consists of a variety of routes, conditions, and scenarios, town 10HD had a simpler layout but offered a high-definition video feed. This meant there were no predefined scenarios or routes present in the CBS2 assets. Constructing a scenario and route for a sound driving episode involved leveraging several aspects:

- Weather Conditions: The weather parameters were set to create consistent and clear environmental conditions to ensure reliable data collection. The chosen settings (e.g., no cloudiness, precipitation, or fog) provided optimal visibility and minimal environmental noise.
- Waypoints: A series of waypoints were defined to create a comprehensive route through Town10HD. These waypoints guided the vehicle along the desired path, covering various areas within the town to capture diverse driving scenarios. Figure 4.4 visualizes the route by overlaying the waypoints on the town map.

- Ego Vehicle: The main vehicle used in the scenario is a Lincoln MKZ 2017, which starts at specific coordinates and is not on autopilot. This allows for controlled testing and data collection.
- Other Actors: The scenario includes other vehicles, such as an Audi TT and a Tesla Model 3, both set on autopilot. These vehicles introduce dynamic elements to the scenario, simulating realistic traffic conditions.

```
<?xml version="1.0" encoding="UTF-8"?>
<routes>
<route id="0" town="Town10HD">
  <weather
     cloudiness="0"
     precipitation="0"
     precipitation_deposits="0"
     wind_intensity="0"
     sun_azimuth_angle="0"
     sun_altitude_angle="70"
     fog_density="0"
     fog_distance="0"
     wetness="0"
  />
  <waypoint pitch="0.0" roll="0.0" x="338.7" y="226.75" yaw="270.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="321.98" y="194.67" yaw="180.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="283.69" y="194.78" yaw="180.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="108.05" y="195.29" yaw="180.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="88.40" y="210.57" yaw="90.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="88.41" y="309.63" yaw="90.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="75.58" y="326.30" yaw="180.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="14.33" y="326.26" yaw="180.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="1.87" y="299.43" yaw="270.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="1.61" y="170.71" yaw="270.0" z="0.0" />
  <waypoint pitch="0.0" roll="0.0" x="1.36" y="47.93" yaw="270.0" z="0.0" />
</route>
</routes>
```

By carefully defining these scenarios, including the specific positions and behaviors of each vehicle, it was ensured that the simulation provided a realistic and controlled environment for data collection. The high-definition video feed from Town 10HD, combined with the dynamic interaction of multiple vehicles, created a rich dataset for training and evaluating the stego model.

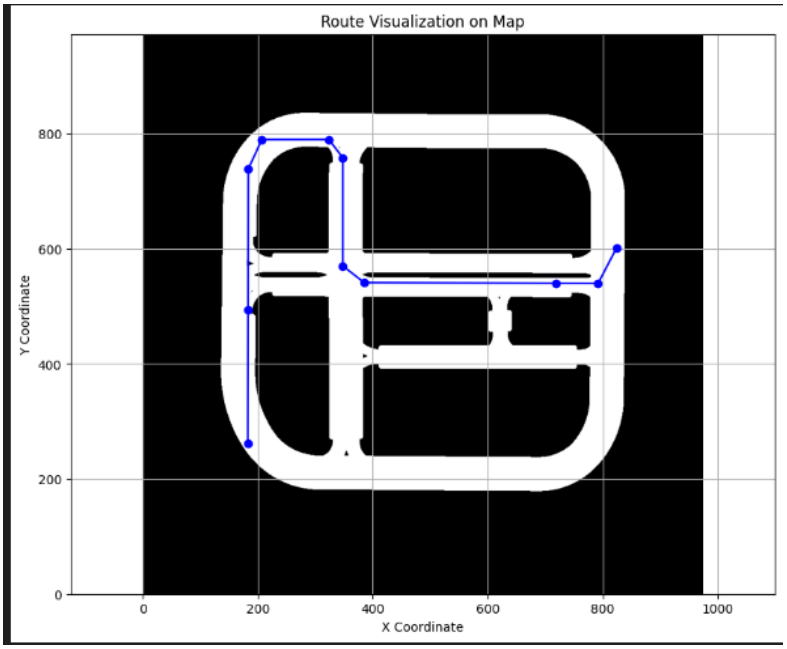


Figure 4.4: Route defined by waypoints overlaying Town 10HD map.

4.3 STEGO

The STEGO repository provided models trained on common datasets such as COCO-Stuff, Cityscapes, and Potsdam, all of which consist of urban environments. This alignment with urban settings made STEGO a compelling choice for the chosen dataset. Utilizing these pre-trained models was a beneficial starting point to understand the model's mechanics and evaluate its performance in similar environments.

Although the STEGO repository included a module for training on custom datasets, it was basic and required significant adjustments to fit the Carla data. Many sections were either missing or incorrect, necessitating extensive revisions of the dataset loader, cropping, and training scripts. These modifications were essential to ensure compatibility and optimal performance with the Carla dataset

4.3.1 Configurations

Before one could start on the dataloading or -cropping, the corresponding config yaml file needed to be set. The main points to focus on are:

Listing 4.1: Configuration YAML file for STEGO model

```
output_root: 'logs'
pytorch_data_dir: '/path/to/your/data/'
experiment_name: "custom_experiment_1"
log_dir: "stego_logs"
azureml_logging: False
submitting_to_aml: False

# Loader parameters
num_workers: 32
max_steps: 15000
batch_size: 64

num_neighbors: 7
dataset_name: "directory"

Used if dataset_name is "directory"
dir_dataset_name: "directory_five_crop_0.5"
dir_dataset_n_classes: 20

has_labels: False
crop_type: "five"
crop_ratio: 0.5
res: 200
loader_crop_type: "center"
```

This configuration file outlines the settings needed for the STEGO model to process the Carla dataset. Key parameters include the output directory, the number of workers, the maximum steps, the batch size, and specific details related to the dataset and cropping methods. Setting these parameters correctly is crucial for efficient data loading and training processes. Incorrect configurations can lead to errors that are not always easy to diagnose.

4.3.2 Cropping Utility

The cropping utility provided by the STEGO repository offers a variety of tools to augment the dataset and improve spatial resolution. These tools are crucial for enhancing the model's ability to generalize across different scenarios by providing diverse and representative data samples. However, working with a custom dataset introduced specific challenges related to image dimensions and shapes, necessitating modifications to the cropping script to ensure compatibility and optimal performance.

Five Crop Utility

The Five Crop utility in the STEGO repository is designed to create augmented versions of images by generating five distinct crops: four from the corners and one from the center. This method is particularly useful for increasing the diversity of the training data without requiring additional raw data. The five crops ensure that the model is exposed to various parts of the image, helping it to learn more robust features. Here's a brief overview of how the Five Crop utility works. For a visualization see figure 4.5:

- **Corner Crops**: The utility extracts four smaller images from the corners of the original image.
- **Center Crop**: In addition to the corner crops, a crop is taken from the center of the image.
- **Maintaining Aspect Ratio**: The crops maintain the aspect ratio of the original image, which helps in preserving the spatial relationships within the image.

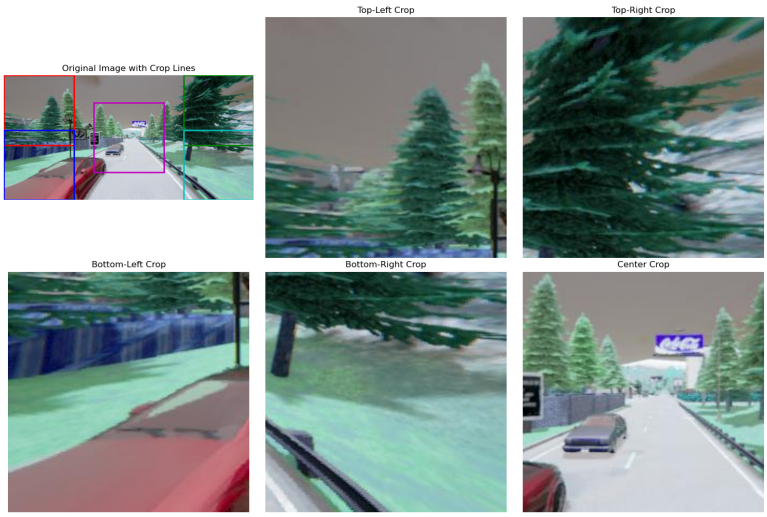


Figure 4.5: Visualization of the Five Crop utility

Modifications for Custom Dataset

Using the Five Crop utility with the Carla dataset presented unique challenges due to the specific dimensions and shapes of the images in this dataset. The

original cropping script in the STEGO repository was designed with standard datasets in mind, which often have consistent image dimensions and formats. The Carla dataset, however, had images with different dimensions and formats, leading to errors and inefficiencies when using the default cropping utility. These issues included:

- **Dimension Mismatches**: The Carla images did not always match the expected input dimensions for the Five Crop utility, resulting in errors during the cropping process.
- **Shape Inconsistencies**: Some images had additional channels or metadata that were not handled by the default script.

To address these issues, significant modifications were made to the cropping script. The main changes included:

- **Dynamic Dimension Handling**: The script was updated to dynamically adjust to the dimensions of the input images, ensuring that the cropping operations could proceed without errors.
- **Shape Normalization**: Additional preprocessing steps were introduced to normalize the shapes of the images, stripping out any unnecessary channels or metadata and converting the images into a consistent format suitable for cropping.

Information Loss

While cropping can lead to potential information loss, the Five Crop utility and the modifications made ensure that diverse and crucial parts of the image are preserved and utilized. By exposing the model to various perspectives within each image, the training data becomes more robust. Additionally, the use of larger datasets and various augmentation techniques further mitigate the risks associated with information loss, making the approach effective for custom data like the Carla dataset.

Here the CARLA_COLOR_TO_CLASS is a custom conversion dictionary that is based on the semantic segmentation label colors provided by the Carla Documentation. Each color is associated with an object:

Listing 4.2: Carla color conversion

```
CARLA_COLOR_TO_CLASS = {
    (0, 0, 0): 0,              # Unlabeled
```

```
    (70, 70, 70): 1,        # Building
    (100, 40, 40): 2,       # Fence
    (55, 90, 80): 3,        # Other
    (220, 20, 60): 4,       # Pedestrian
    (153, 153, 153): 5,     # Pole
    (157, 234, 50): 6,      # RoadLine
    (128, 64, 128): 7,      # Road
    (244, 35, 232): 8,      # Sidewalk
    (107, 142, 35): 9,      # Vegetation
    (0, 0, 142): 10,        # Vehicle
    (102, 102, 156): 11,    # Wall
    (220, 220, 0): 12,      # TrafficSign
    (70, 130, 180): 13,     # Sky
    (81, 0, 81): 14,        # Ground
    (150, 100, 100): 15,    # Bridge
    (230, 150, 140): 16,    # RailTrack
    (180, 165, 180): 17,    # GuardRail
    (250, 170, 30): 18,     # TrafficLight
    (110, 190, 160): 19,    # Static
    (170, 120, 50): 20,     # Dynamic
    (45, 60, 150): 21,      # Water
    (145, 170, 100): 22     # Terrain
}
```

The modification ensures that label images are converted to class colors only when the 'cropping' flag is set to 'False'. This allows the cropping process to work with the original label images, avoiding potential errors and maintaining label integrity during cropping. After cropping, adjust the configuration YAML file to set the dataset path to the cropped dataset before proceeding to the next step(s).

4.3.3 Precompute KNN indices

The STEGO algorithm leverages the concept of k-nearest neighbors (k-NNs) to enhance the feature consistency and efficiency of the model. The precomputation of k-NNs is an essential step that significantly contributes to the overall performance of the segmentation model. This process is detailed in the STEGO repository within the `precompute_knns.py` script.

Purpose and Importance

The k-NN precomputation serves several critical purposes:

- **Feature Consistency**: Precomputing k-NNs ensures that similar features are grouped together, facilitating the learning of robust feature representations. This is vital for achieving high-quality segmentation results.
- **Efficiency**: Calculating k-NNs for all features on-the-fly during training or inference would be computationally prohibitive. By precomputing these neighbors, the algorithm can perform more efficiently, as the necessary data is readily available.
- **Graph Construction**: The k-NNs are used to construct a feature graph, where nodes represent different image regions and edges denote the similarity between these regions. This graph structure is crucial for the energy-based optimization process that STEGO employs.

Implementation

The `precompute_knns.py` script in the STEGO repository handles this precomputation. The script follows these steps:

1. **Feature Extraction**: Using a pre-trained model like DINO, the script extracts normalized feature vectors from the input images.
2. **k-NN Computation**: It calculates the pairwise similarities between these feature vectors using dot products. The top-k nearest neighbors for each vector are identified and stored.
3. **Data Handling**: The script supports various datasets and crop types, ensuring that the k-NNs are computed and cached appropriately for future use. This caching significantly speeds up the subsequent training and inference stages.

4.3.4 Training Phase

The training phase in STEGO involves several steps to ensure that the model learns to segment images effectively. Below is a detailed breakdown of the key steps and parameters involved:

The training script, `train_segmentation.py`, orchestrates the training process. Here's a breakdown of its main components and functions:

- **Data Loading**: The script utilizes the `ContrastiveSegDataset` class to load the training and validation datasets. This class supports various augmentations to enhance the training data.

Model Params	Feature Contrastive Params	Weights
extra_clusters: 0	pointwise: True	neg_inter_weight: 0.75
use_true_labels: False	feature_samples: 30	pos_inter_weight: 0.35
use_recalibrator: False	neg_samples: 10	pos_intra_weight: 0.75
model_type: "vit_small"	aug_alignment_weight: 0.0	neg_inter_shift: 0.30
arch: "dino"	correspondence_weight: 1.2	pos_inter_shift: 0.10
use_fit_model: True	**CRF Params**	pos_intra_shift: 0.15
dino_feat_type: "feat"	crf_weight: 0.0	rec_weight: 0.1
projection_type: "nonlinear"	alpha: 0.5	repulsion_weight: 0.05
dino_patch_size: 8	beta: 0.15	
granularity: 1	gamma: 0.05	
continuous: True	w1: 10.0	
dim: 64	w2: 3.0	
dropout: True	shift: 0.00	
zero_clamp: True	crf_samples: 1000	
	color_space: "rgb"	

Table 4.1: Model Parameters, Feature Contrastive Parameters, and Weights

- **Model Initialization**: The `LitUnsupervisedSegmenter` class initializes the segmentation model. Depending on the architecture specified in the configuration (e.g., DINO), different feature extraction and segmentation strategies are used.
- **Loss Functions**: Several loss functions are employed to guide the training:
  - **Contrastive Loss**: Ensures that similar features are closer together in the feature space.
  - **CRF Loss**: (if enabled) Enhances the segmentation boundaries using Conditional Random Fields.
  - **Reconstruction Loss**: Ensures the features can be reconstructed from the latent space effectively.
- **Training Loop**: The training loop iterates over the dataset, computing the losses and updating the model weights using the specified optimizers. It also logs metrics and checkpoints the model periodically.
- **Validation**: The model's performance is evaluated on the validation set at regular intervals to monitor overfitting and ensure generalization.

4.4 Evaluation

4.4.1 Metrics

During the training and evaluation of the STEGO model, several metrics are logged to monitor the performance and progress of the model. These metrics are crucial for understanding how well the model is learning and how effectively it is segmenting the images. In the appendix A.1 are the key evaluation metrics denoted, along with explanations of how they work and what the ideal values would be. By logging and monitoring these metrics, I can gain a comprehensive understanding of the model's performance and identify areas that may need improvement. Ideally, as training progresses, the losses should decrease, and the accuracy and distance metrics should show favorable trends, indicating that the model is learning effectively and segmenting the images accurately. For the results for these metrics refer to chapter 5 Results.

4.4.2 Suggested Approaches

Early on, there were issues regarding cluster loss and its accompanying metrics: e.g., starting below zero, inverse trends, etc. To address these issues, several approaches were implemented. Although there was no significant change in the results, the details of these approaches are provided in appendix A.2. For their reasoning, see Chapter 6.

4.4.3 Real-Time Segmentation

To achieve real-time segmentation of video feeds using the STEGO algorithm, several approaches and optimizations were explored. This section outlines the steps taken, the rationale behind each method, and the challenges encountered.

Initial Setup

The STEGO model trained in the previous step was loaded from a checkpoint. The model was initially configured to run on a GPU to exploit faster computation times.

```
# Paths and model loading
dir = "path/to/trained/model"
sav_model = "saved_model.ckpt"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load the PyTorch model
```

```
model = LitUnsupervisedSegmenter.load_from_checkpoint(join(dir, sav_model)) \
.to(device).eval().half()
```

Sequential Approaches

For a more elaborate depiction of the code implemented for this section see appendix.

1. **Frame-by-Frame Processing on GPU**
    - **Goal:** Utilize GPU for faster processing by handling each video frame individually.
    - **Method:** Each frame was processed sequentially using the GPU.
    - **Issue:** Encountered *CUDA out of memory* errors due to the high memory demand of processing high-resolution video frames individually.
2. **Frame-by-Frame Processing on CPU**
    - **Goal:** Circumvent GPU memory limitations by processing frames on the CPU.
    - **Method:** The model and frame processing were moved to the CPU.
    - **Issue:** Processing on the CPU was significantly slower than on the GPU, rendering real-time performance unfeasible.
3. **Batch Processing on GPU**
    - **Goal:** Improve memory management and reduce overhead by processing multiple frames in batches.
    - **Method:** Frames were collected into batches and processed collectively on the GPU.
    - **Issue:** Continued to experience *CUDA out of memory* errors, indicating that the batch size was still too large for the available GPU memory.
4. **Mixed Precision and Smaller Batch Size**
    - **Goal:** Reduce memory usage further and fit within GPU memory constraints.
    - **Method:**
        - Implemented mixed precision using `torch.cuda.amp`.
        - Reduced the batch size.

- **Tools:** `torch.cuda.amp` for mixed precision.
- **Outcome:** Improved memory management and reduced *CUDA out of memory* errors, but adjustments were necessary based on the GPU's memory capacity.

5. **Final Implementation**

   - **Approach:** The final implementation combined batch processing with mixed precision and efficient memory management techniques. Specific adjustments included:
     - **Path and Model Loading:** The model was loaded on the GPU and configured to run in half precision.

       ```
       device = torch.device("cuda" if torch.cuda.is_available() \
       else "cpu")

       model = LitUnsupervisedSegmenter.load_from_checkpoint(join(dir, \
       sav_model)).to(device).eval().half()
       ```

     - **Batch Processing:** Frames were processed in batches with a reduced batch size of 4.

       ```
       BATCH_SIZE = 4
       def process_batch_with_stego(frames, model, use_linear_probe=True):
           # Batch processing implementation
           ...
       ```

     - **Real-Time Display:** Processed frames were displayed side-by-side with the original frames for real-time visualization.

       ```
       def display_segmented_frames():
           while True:
               frame_pair = segmented_frame_queue.get()
               if frame_pair is None:
                   break
               original_frame, segmented_frame = frame_pair
               combined_frame = np.hstack((original_frame, \
               segmented_frame)).astype(np.uint8)

               cv2.imshow('Segmented Frame', combined_frame)
               if cv2.waitKey(1) & 0xFF == ord('q'):
                   break
           cv2.destroyAllWindows()
       ```

– **Queue Management:** Utilized queues to manage frames efficiently between reading, processing, and displaying stages.

```
frame_queue = Queue(maxsize=100)
segmented_frame_queue = Queue(maxsize=100)

def read_frames():
    # Reading frames from video
    ...

def process_frames():
    # Processing frames in batches
    ...
```

Steps to Monitor and Manage GPU Resources

- **Use `nvidia-smi`:** Monitor GPU usage in real-time to understand memory usage and identify bottlenecks.
- **Reduce Batch Size:** Adjust batch size to fit within the available GPU memory.
- **Mixed Precision:** Ensure mixed precision is correctly enabled to utilize the reduced memory usage.

Testing Video

For the segmented test video please refer to the repository corresponding to this thesis. For a snippet see appendix the example segmented video on the repository of this thesis.

Chapter 5

Results

The results of the training phase, and the corresponding evaluation metrics, the segmented testing video and the real-time segmentation are presented here.

5.1 Training Losses

The provided images and graphs from the evaluation process include the following:

5.1.1 Segmentation with Labels

The images in figure 5.1 show the original frames, ground truth labels, linear probe predictions, and cluster probe predictions.

Visual inspection of these images reveals that STEGO can segment different objects and scenes somewhat accurately. However, the linear probe classification yields better results than the cluster probe classification, as indicated by the more accurate color and label designations in the linear probe. The cluster probe, on the other hand, shows some incorrect color/label assignments. It is important to note that these images are snapshots representing intermittent progress at a specific epoch/step. The overall segmentation quality is more visually coherent in the full test segmentation video, where the distinctions between segmented objects (in one image/frame) are clearer. To paint a better picture a segmented (whole) image is shown in figure 5.2 and 5.3. The former showing remarkable performance in segmenting the objects in an image while also classifying them correctly using the linear classifier (lower left: linear probe, upper right: cluster probe). You can clearly see the cars with proper outline as well as correct color label. While the cluster probe segments in a similar fashion, the (in)ability to correctly assign the color label to these segments is a concern that should be looked into for future work. See section 6 for tried methods and evaluation. The latter figure depicts that

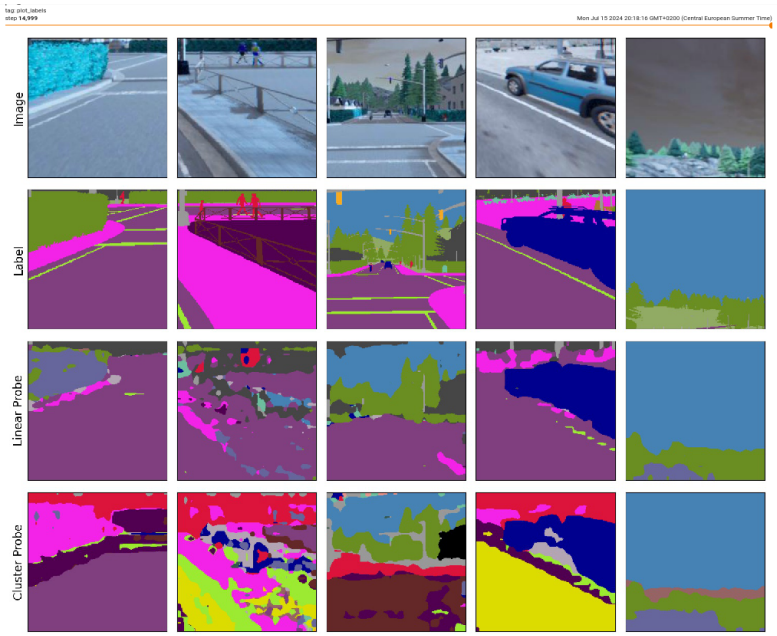


Figure 5.1: Segmented Training snippet with labels

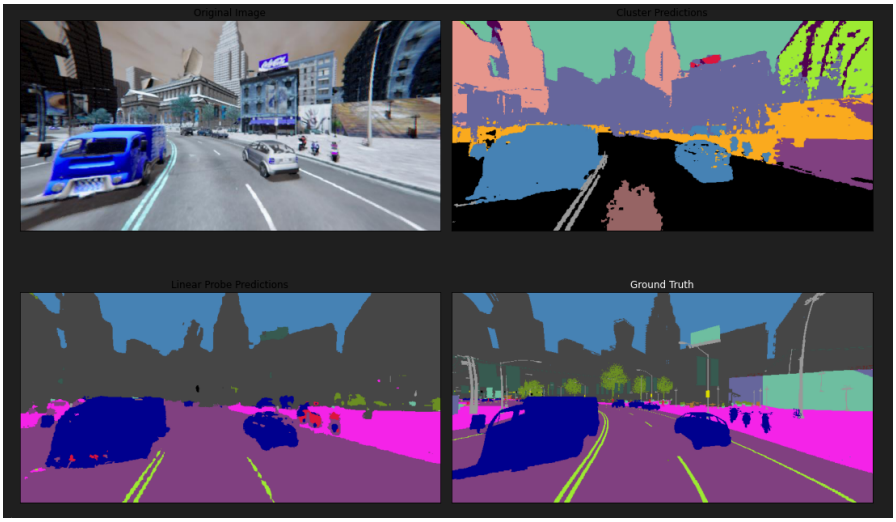


Figure 5.2: Zoomed in Segmented snippet close by

after a few 'meter' in front of the car the model loses segmentation performance. A possible bottleneck could be the resolution, as fewer pixels present means less distinction.

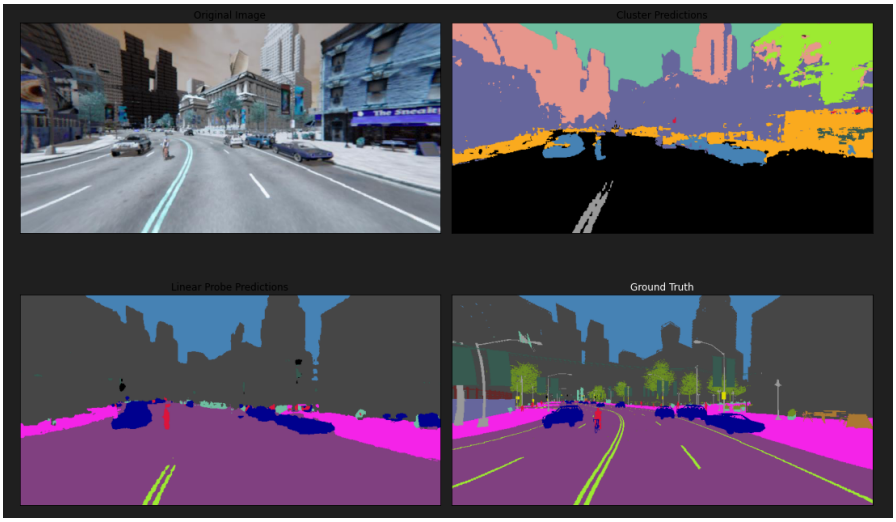
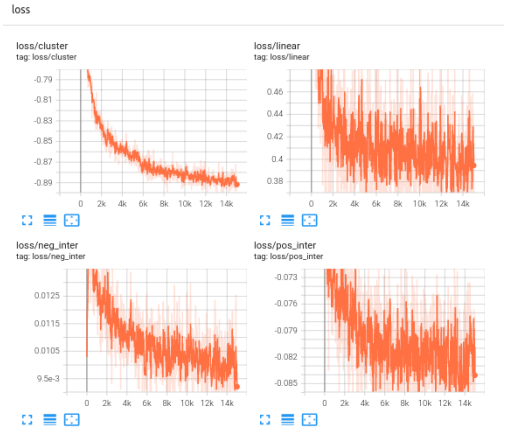


Figure 5.3: Zoomed in Segmented snippet

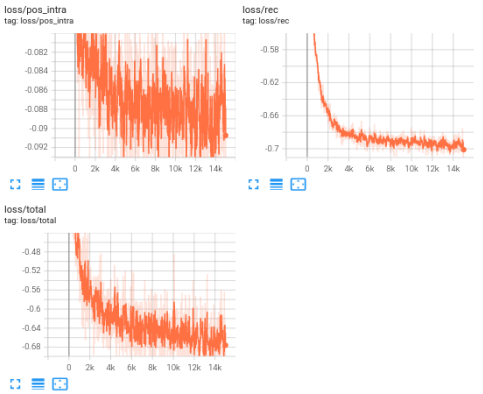
5.1.2 Loss Metrics

The loss graphs illustrate the training and evaluation loss over iterations for various components, providing insight into the model's optimization process.

- **loss/cluster:** This loss behaves as expected for a loss function, generally decreasing over time. However, it starts below zero, which is unusual and might indicate an initial misconfiguration in the cluster center settings or the normalization process.
- **loss/linear:** The linear loss shows a decreasing trend but with significant oscillations. These oscillations suggest that the model parameters are undergoing substantial adjustments, which might be a result of high learning rates or the complexity of the data.
- **loss/neg_inter:** This loss component spikes initially but gradually decreases over time. The initial spike might be due to the model adjusting to the inter-class variance, and the subsequent decrease indicates improved separation between different classes.
- **loss/pos_inter:** The positive inter-class loss decreases over time but with heavy oscillations. Starting below zero, this behavior might again point to initial configuration issues or challenges in maintaining inter-class compactness.
- **loss/pos_intra:** Starting below zero and exhibiting slight decreases with heavy oscillations, this loss suggests difficulties in achieving intra-class com-



(a) Cluster loss, Linear loss, Negative Inter-class (neg_inter) loss, and Positive Inter-class (pos_inter) loss.



(b) Positive Intra-class (pos_intra) loss, Reconstruction (Rec) loss, and Total loss.

Figure 5.4: Loss Metrics showing the training and evaluation loss over iterations for different loss components.

pactness. The heavy oscillations could be due to the variability in how different instances of the same class are being clustered.

- **loss/rec:** The reconstruction loss starts below zero but follows a typical decreasing trend. This indicates that the model is progressively better at reconstructing the input data, despite the unusual initial negative values.
- **loss/total:** The total loss is a combination of all the above components, integrating their behaviors. It reflects the overall optimization process, showing how the model is balancing various aspects of the segmentation task.

The presence of losses starting below zero could suggest an issue with the initial setup of the model, such as incorrect initialization of cluster centers or improper normalization. Addressing these initial settings might lead to more stable and expected loss behaviors throughout the training process.

5.1.3 Compactness Deviation (CD) Metrics

The CD metrics for negative inter-class, positive inter-class, and positive intra-class deviations show the variance in compactness of the feature clusters.

The observed trends are as follows:

- **cd/neg_inter:** This metric shows an initial spike up, followed by a quick decrease over time with some oscillation. The initial spike indicates a large

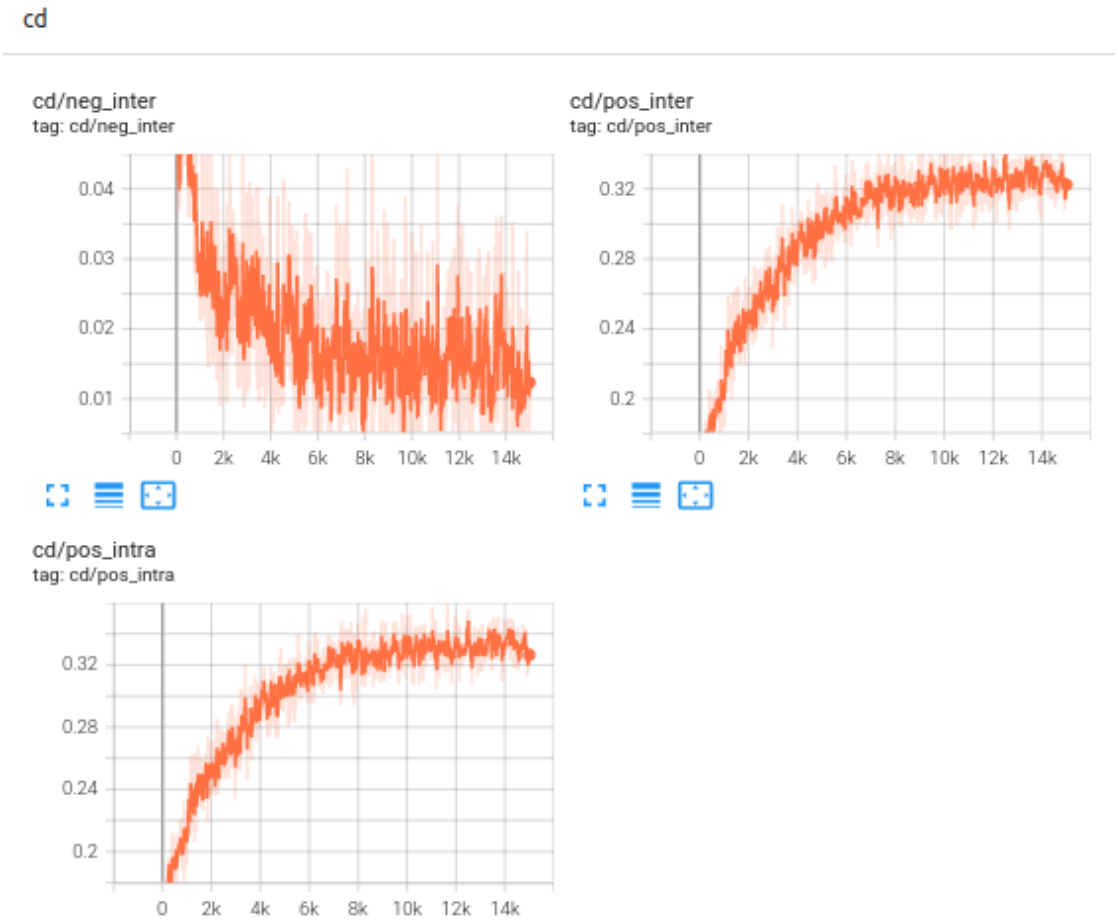


Figure 5.5: Compactness Deviation (CD) Metrics showing the variance in feature cluster compactness.

variance in the inter-class distances early in the training process. The subsequent decrease suggests that the model is becoming better at separating different classes as training progresses. The minor oscillations indicate ongoing adjustments in cluster separation.

- **cd/pos_inter:** This metric increases over time with a decreasing rate. Ideally, one would expect this value to decrease, indicating that features within the same class are becoming more compact and distinct from each other. The observed increase suggests that features within the same class are becoming more spread out, which is not desirable for effective clustering.
- **cd/pos_intra:** Similar to the positive inter-class deviation, this metric also increases over time with a decreasing rate. Again, this is not a desirable

trend as it indicates that features within the same class are becoming less compact. This can lead to poor intra-class compactness and potentially worse segmentation performance.

The increasing trends in **cd/pos_inter** and **cd/pos_intra** are particularly concerning as they suggest that the model is not effectively clustering features within the same class. This could be due to several factors, including improper initialization, inadequate feature learning, or issues with the loss function's ability to enforce compactness. Addressing these issues might involve re-evaluating the initialization of cluster centers, modifying the training process, or adjusting the loss functions to better enforce intra-class compactness.

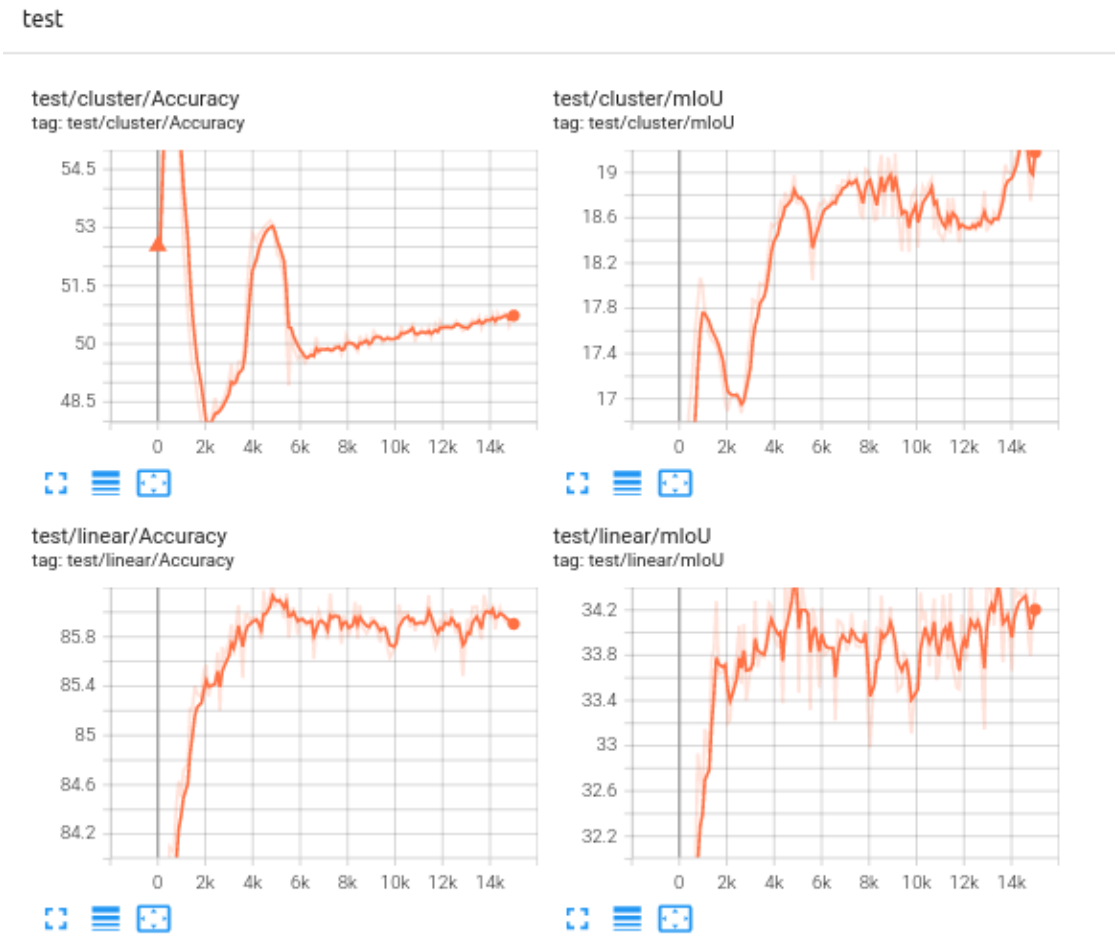


Figure 5.6: Test Metrics showing accuracy and mean Intersection over Union (mIoU) for cluster and linear probes.

5.1.4 Test Metrics

Accuracy and mean Intersection over Union (mIoU) for both cluster and linear probes are presented in the test metrics.

The test metrics provide valuable insights into the model's performance during training. The observed trends for each metric are as follows:

- **cluster/accuracy:** This metric shows a peak around 1,000 steps, after which it decreases quickly to an all-time low of 48%. It then rises to 53% at step 5,000, followed by another decrease to 50% at step 6,000. After this, the accuracy slowly increases. This fluctuation indicates instability in the model's performance in terms of cluster accuracy, possibly due to changes in cluster assignments and adjustments during training.

- **cluster/mIoU:** The mean Intersection over Union (mIoU) for clusters slowly increases over time, with occasional significant dips. It stabilizes around a value of 19 towards the end of the training. The overall increasing trend is positive, indicating improved segmentation performance, but the dips suggest intermittent issues with cluster assignments.

- **linear/accuracy:** The linear accuracy increases with a decreasing rate until step 5,000, reaching 86%. After this peak, it decreases slightly to 85.8% and then oscillates around this value for the remainder of the training. This behavior indicates that the linear probe achieves stable and high accuracy relatively early in the training process, with minor fluctuations around a high value.

- **linear/mIoU:** The mean Intersection over Union (mIoU) for the linear probe follows a similar trend to the linear accuracy, peaking at a value of 34.2. The trend shows an initial increase with a decreasing rate, indicating that the linear probe effectively learns to segment the images with high precision early on, with some minor fluctuations.

The observed trends suggest that while the linear probe demonstrates more stable and higher performance in terms of both accuracy and mIoU, the cluster probe experiences more variability and lower overall performance. The initial peak and subsequent fluctuations in cluster accuracy might be due to the dynamic nature of clustering during training, where cluster centers are adjusted frequently, leading to temporary instability. In contrast, the linear probe benefits from a more straightforward supervised learning approach, resulting in more consistent performance.

5.2 Summary of Findings

In summary, the evaluation of the segmentation method yields the following key insights:

- **Visual Segmentation Quality:** The visual inspections indicate that the linear probe classification consistently provides more accurate and reliable segmentation results compared to the cluster probe. The linear probe's color and label assignments are more precise, highlighting its effectiveness in segmenting and classifying objects correctly.
- **Training Losses:** The analysis of the loss metrics shows expected decreasing trends for most components, though some losses start below zero, indicating potential issues with initial model setup or normalization processes. Oscillations in the linear and inter-class losses suggest adjustments in the model parameters, possibly due to high learning rates or complex data.
- **Compactness Deviation Metrics:** The CD metrics reveal that the model struggles with achieving intra-class compactness and inter-class separation, as indicated by the increasing trends in **cd/pos_inter** and **cd/pos_intra**. These trends suggest potential issues with feature clustering that require further investigation and adjustment. The cluster loss issues present also affect these metrics.
- **Test Metrics:** The test metrics show that while the linear probe achieves high and stable performance in terms of accuracy and mIoU, the cluster probe exhibits more variability and lower overall performance. The initial fluctuations in cluster accuracy highlight the dynamic nature of clustering during training.

5.2.1 Overall Effectiveness:

The results indicate that the method, particularly the linear probe, is effective in segmenting and classifying objects in images with high accuracy and reliability. The linear probe's performance demonstrates the method's potential for practical applications, despite the noted issues with the cluster probe and certain loss metrics. Addressing the identified challenges could further enhance the method's stability and effectiveness.

5.3 Real-Time Segmentation

While the STEGO algorithm demonstrates impressive results in unsupervised semantic segmentation, achieving real-time performance remains challenging due to its computational intensity, post-processing requirements, and hardware limitations. Further optimizations, both algorithmic and hardware-based, are necessary to bridge the gap towards real-time applications.

The implementation of real-time video segmentation using the STEGO algorithm yielded significant insights and performance improvements through various approaches. The results from each method are summarized below.

Frame-by-Frame Processing on GPU

- **Performance:** Initial attempts to process each video frame sequentially on the GPU demonstrated fast processing times for individual frames.
- **Issue:** However, this approach encountered *CUDA out of memory* errors due to the high memory demand when processing high-resolution frames.

Frame-by-Frame Processing on CPU

- **Performance:** Shifting the processing to the CPU circumvented GPU memory limitations but resulted in significantly slower processing times.
- **Issue:** This method was not feasible for real-time performance, as the CPU could not handle the computational load efficiently.

Batch Processing on GPU

- **Performance:** Processing frames in batches aimed to improve memory management and reduce overhead.
- **Issue:** Despite this, *CUDA out of memory* errors persisted, indicating the batch size was too large for the available GPU memory.

Mixed Precision and Smaller Batch Size

- **Performance:** Implementing mixed precision with `torch.cuda.amp` and reducing the batch size improved memory management.
- **Outcome:** This approach successfully reduced *CUDA out of memory* errors, though careful adjustments were necessary based on the GPU's memory capacity.

Final Implementation

- **Batch Processing with Mixed Precision:** The final implementation utilized batch processing combined with mixed precision and efficient memory management techniques to achieve stable performance without memory issues.
- **Real-Time Display:** Processed frames were displayed side-by-side with the original frames, enabling real-time visualization and verification of segmentation results.
- **Queue Management:** Queues were employed to efficiently manage frames between reading, processing, and displaying stages, ensuring smooth real-time processing.
- **Performance:** This approach improved segmentation performance and handled the video feed more efficiently with the available GPU resources, but real-time segmentation was not fully achieved.

Chapter 6

Discussion/Future Work

This chapter provides a comprehensive discussion of the results presented in the previous sections, reflecting on the effectiveness of the methodology and offering insights into potential improvements. The primary focus is on evaluating the performance of the STEGO algorithm for real-time semantic segmentation in autonomous driving scenarios simulated using the CARLA environment. Key aspects include data collection, training outcomes, and real-time segmentation performance.

6.1 Data Collection and Preparation

The data collection process was crucial in ensuring the robustness of the STEGO network. By leveraging the CARLA simulator, we were able to generate diverse datasets that mimic real-world driving conditions. The use of Town 04 for training provided a rich variety of environments, including urban, highway, and outdoor scenes, which were instrumental in training a model capable of handling diverse driving scenarios. However, issues with augmentation settings in the initial data collection led to faulty data, necessitating a meticulous logging function to ensure the integrity of the dataset.

The transition from LMDB format to the specific format required by the STEGO model posed significant challenges. The implementation of a function to convert and preprocess the data, along with parallel processing for larger datasets, proved effective in managing computational resources. This process included dynamically adjusting to image dimensions, normalizing shapes, and ensuring efficient memory usage through garbage collection and error handling mechanisms. These steps ensured the integrity and usability of the data, but highlight an area for potential improvement, particularly in automating and optimizing data conversion processes.

6.2 Training Outcomes

The training phase of the STEGO network revealed several key insights:

- **Loss Metrics:** The loss metrics provided a detailed view of the model's optimization process. While most loss components behaved as expected, starting below zero for several metrics (e.g., cluster loss, positive inter-class loss) suggested issues with initial settings or normalization processes. These anomalies need to be addressed to stabilize the training process from the outset. Possbible issues could also orginate from the precompute of the knn indices, where the label color classes are incorrect handled.
- **Compactness Deviation (CD) Metrics:** The CD metrics showed trends that were not entirely favorable. While negative inter-class deviation decreased over time, both positive inter-class and intra-class deviations increased, indicating that features within the same class became more spread out over time. This spread suggests that the model struggled to maintain compact clusters, affecting segmentation performance. Future work could involve refining the initialization of cluster centers and enhancing loss functions to enforce better intra-class compactness.
- **Test Metrics:** The test metrics revealed a clear disparity between the cluster and linear probes. The linear probe demonstrated higher and more stable performance in both accuracy and mIoU, while the cluster probe performed poorly, indicating significant room for improvement in the clustering approach. This suggests that the clustering method requires further refinement and possibly the integration of more sophisticated clustering techniques.

6.3 Approaches for Cluster Issues

Several approaches were attempted to address the issues with the cluster loss, but they did not result in significant improvements. Here, I discuss the impact and interpretation of each approach, explained in appendix A.2.

6.3.1 L2 Regularization

L2 regularization was applied to prevent overfitting by adding a penalty to the loss function proportional to the sum of the squared values of the model parameters. While L2 regularization is a standard technique to prevent overfitting, in this case, it did not yield significant improvements in the cluster

loss. This could be due to the specific nature of the clustering problem where the regularization strength might need further tuning. Additionally, the regularization may not address the core issues of clustering dynamics in the feature space.

6.3.2 Warm-Up Stage for Cluster Loss

A warm-up stage was implemented to gradually increase the influence of the cluster loss during the initial training steps. This was expected to stabilize the training process by allowing the model to learn robust features first. However, the warm-up approach did not lead to significant improvements, suggesting that the problem might lie in the inherent difficulty of the clustering task or the specific implementation of the cluster loss. Future work could explore different warm-up durations or more gradual increases in the cluster loss weight.

6.3.3 Cluster Initialization Methods

Different initialization strategies, including Xavier and uniform initialization, were tested for the cluster centroids. These methods aimed to find a better starting point for the clustering process. Despite these efforts, no significant improvement was observed, indicating that the initialization method alone may not be sufficient to enhance clustering performance. This suggests the need for more sophisticated initialization strategies or adaptive methods that can dynamically adjust cluster centers during training.

6.3.4 Positive Offset in Cluster Loss

Adding a positive offset to the cluster loss was intended to prevent initial loss values from being too small, thereby promoting better gradient flow. Despite this adjustment, the results did not improve significantly. This might be due to the offset not being large enough to make a meaningful impact, or because the issue lies deeper in the loss function's formulation, knn precomputation or the clustering mechanism itself. Further experimentation with different offset values or alternative formulations could be beneficial.

6.3.5 Entropy-Based Loss Function

An entropy-based loss function was implemented to encourage a more uniform distribution of cluster assignments, preventing collapse to a few clusters. Although theoretically sound, this approach initially resulted in NaN values for the cluster loss, indicating numerical instability or implementation issues. This highlights the challenge of implementing entropy-based methods in practice. Future work should focus on stabilizing this loss function, potentially through techniques like gradient clipping, regularization, or better numerical handling.

6.3.6 Future Application

To improve clustering performance and overall segmentation quality, future work should consider the following areas:

- **Refinement of Loss Functions:** Further refinement and testing of loss functions, particularly entropy-based and other advanced loss formulations, to achieve more stable and effective training.
- **Advanced Initialization Techniques:** Exploring more advanced initialization techniques, potentially leveraging pre-trained models or data-driven initialization strategies.
- **Enhanced Data Augmentation:** Improving data augmentation techniques to ensure better feature learning and generalization.
- **Regularization and Optimization:** Tuning regularization parameters and optimization techniques to better suit the specific challenges of clustering in semantic segmentation.
- **Integration of Sophisticated Clustering Methods:** Integrating more sophisticated clustering methods, such as spectral clustering or graph-based clustering approaches, to enhance cluster quality.

By addressing these areas, future iterations of the STEGO algorithm can achieve better clustering performance and overall improvements in semantic segmentation tasks.

6.4 Real-Time Segmentation Performance

Achieving real-time segmentation was a critical, although secondary, goal of this research. The initial goal was to achieve a somewhat accurate unsupervised seg-

mentation. Several approaches were tested, each with its own set of challenges and outcomes:

- **Frame-by-Frame Processing:** Both GPU and CPU approaches for processing frames individually faced significant limitations. The GPU approach was hindered by memory constraints, leading to *CUDA out of memory* errors, while the CPU approach was too slow for real-time applications.
- **Batch Processing:** Batch processing on the GPU showed promise in managing memory more effectively, but still encountered memory limitations with larger batch sizes. The introduction of mixed precision processing helped mitigate some of these issues by reducing memory usage, but required careful management of batch sizes to avoid memory overflows.
- **Final Implementation:** Despite efforts to combine batch processing, mixed precision, and efficient queue management, the desired real-time segmentation performance was not achieved. This persistent challenge suggests that the bottleneck could also lie elsewhere, possibly in the algorithm's inherent complexity or in the need for further optimization at the hardware level, or the constraints opencv places on the system.

6.5 Reflections and Improvements

- **Initialization and Normalization:** Addressing the initial negative values in loss metrics through better initialization and normalization techniques could stabilize the training process and improve convergence rates.
- **Cluster Compactness:** Enhancing the clustering approach to maintain tighter intra-class compactness and better inter-class separation is critical. This could involve exploring alternative clustering algorithms or incorporating additional constraints in the loss functions to enforce compactness.
- **Real-Time Processing:** The failure to achieve real-time performance indicates a need for developing more efficient algorithms that inherently reduce computational and memory overhead. Future work could focus on advanced parallel processing techniques or hardware acceleration to meet the stringent requirements of real-time applications.
- **Data Augmentation:** Ensuring the integrity and diversity of training data is essential. Automating the data collection and preprocessing pipeline to handle different environments and scenarios more efficiently could enhance the model's robustness and generalization capabilities.

- **Algorithmic Refinements:** Exploring newer architectures and methodologies that balance accuracy and efficiency better than the current implementation could lead to more robust real-time segmentation models. Techniques such as lightweight neural networks, model pruning, and quantization could be investigated to reduce computational demands.

6.6 Future Work

The methodology and results presented in this thesis lay a strong foundation for real-time semantic segmentation in autonomous driving applications. Although the current implementation shows promising results, further refinements and optimizations are essential to fully harness the potential of these technologies in real-world scenarios. The insights gained from this research open avenues for future advancements, contributing to the development of more accurate, efficient, and reliable autonomous driving systems. One potential direction for future work is to implement and compare a different segmentation model. An initial setup for an alternative model was explored, but time constraints prevented achieving a workable outcome. Details of this explored unsupervised segmentation model can be found in the readme section in the appendix **??** and on the repository. While there is an initial implementation, no tests or results have been conducted yet. Another direction could be to further fine-tune the current approach and then train the driving agent to evaluate improvements in the self-driving environment with that implementation. This was beyond the scope of this thesis, but if given the opportunity, I would like to continue working on this approach.

Chapter 7

Conclusion

In this thesis, the application of unsupervised semantic segmentation for autonomous driving cars was explored, specifically focusing on leveraging the CARLA simulator and state-of-the-art segmentation algorithms. This approach integrated techniques from existing methods like Learning by Cheating (LBC), World on Rails (WoR), and Cheating by Segmentation (CBS), with enhancements for real-time performance and improved accuracy.

This research demonstrated the feasibility of using unsupervised learning techniques for semantic segmentation in dynamic driving environments. The implementation of the Self-Supervised Transformer with Energy-based Graph Optimization (STEGO) showed promising results, particularly when combined with advanced data collection and preprocessing strategies. However, achieving real-time performance remains a significant challenge due to the computational intensity of the algorithm and the need for further optimization.

Bibliography

[1] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[2] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE international conference on computer vision*, pages 2722–2730, 2015.

[3] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15590–15599, 2021.

[4] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[6] Mark Hamilton, Zhoutong Zhang, Bharath Hariharan, Noah Snavely, and William T Freeman. Unsupervised semantic segmentation by distilling feature correspondences. *arXiv preprint arXiv:2203.08414*, 2022.

[7] Rasheed Hussain and Sherali Zeadally. Autonomous cars: Research results, issues, and future challenges. *IEEE Communications Surveys Tutorials*, 21(2):1275–1313, 2019.

[8] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[9] Mong H Ng, Kaahan Radia, Jianfei Chen, Dequan Wang, Ionel Gog, and Joseph E Gonzalez. Bev-seg: Bird's eye view semantic segmentation using geometry and semantic point cloud. *arXiv preprint arXiv:2006.11436*, 2020.

[10] Thomas van Orden. Cheating by segmentation. Bachelor thesis, Universiteit van Amsterdam, 2021.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[12] Narek Tumanyan, Omer Bar-Tal, Shai Bagon, and Tali Dekel. Splicing vit features for semantic appearance transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10748–10757, 2022.

[13] Maël Wildi. Conditional imitation learning with pyramid perception modules. Master's thesis, École Polytechnique Fédérale de Lausanne, 2022.

[14] Runsheng Xu, Zhengzhong Tu, Hao Xiang, Wei Shao, Bolei Zhou, and Jiaqi Ma. Cobevt: Cooperative bird's eye view semantic segmentation with sparse transformers. *arXiv preprint arXiv:2207.02202*, 2022.

[15] Wenjia Yang, Shuhan Wang, and Junchi Jiang. Adaptive semantics-oriented multimodal object detection for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1404–1413, 2021.

[16] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6), 2021.

Appendix A

Appendix A

A.1 Methodology Metrics

A.1.1 Loss Components

- **Total Loss**: The sum of all individual losses. It represents the overall objective that the model is optimizing. A lower total loss indicates better overall performance. Ideally, this value should decrease and stabilize as training progresses.
- **Linear Probe Loss**: This loss measures how well the features can predict the actual labels using a linear classifier. It is computed using a cross-entropy loss function. A lower linear probe loss indicates better predictive performance of the linear probe. Ideally, this value should decrease steadily.
- **Cluster Loss**: This loss measures the effectiveness of the clustering probe in grouping similar features together. A lower cluster loss indicates better clustering performance. Ideally, this value should decrease and stabilize.
- **Contrastive Correlation Losses**:
  - **Positive Intra-cluster Loss**: Measures the distance between features within the same cluster. Lower values indicate that the features within the same cluster are closer together.
  - **Positive Inter-cluster Loss**: Measures the distance between features in different clusters. Higher values indicate that the features in different clusters are well-separated.
  - **Negative Inter-cluster Loss**: Measures the distance between negative pairs (features that should not be close). Lower values indicate better separation of negative pairs.

A.1.2 Accuracy Metrics

- **Mean Intersection over Union (mIoU)**: mIoU is a common evaluation metric for image segmentation tasks. It calculates the intersection over union for each class and then averages it. A higher mIoU indicates better segmentation performance. Ideally, this value should increase as training progresses.
- **Pixel Accuracy**: The ratio of correctly classified pixels to the total number of pixels. A higher pixel accuracy indicates better segmentation performance. Ideally, this value should also increase as training progresses.
- **Cluster Accuracy**: The accuracy of the clustering probe in assigning the correct labels to the features. A higher cluster accuracy indicates better clustering performance. Ideally, this value should increase as training progresses.

A.1.3 Distance Metrics

- **Intra-cluster Distance (cd/pos_intra)**: Measures the average distance between features within the same cluster. Lower values indicate tighter clustering of similar features. Ideally, this value should decrease and stabilize.
- **Inter-cluster Distance (cd/pos_inter)**: Measures the average distance between features in different clusters. Higher values indicate better separation between different clusters. Ideally, this value should increase and stabilize.
- **Negative Inter-cluster Distance (cd/neg_inter)**: Measures the average distance between negative pairs. Lower values indicate better separation of negative pairs. Ideally, this value should decrease and stabilize.

A.1.4 Visual Metrics

- **Label Frequency Histogram**: A visual representation of the frequency of each predicted label. It helps in understanding the distribution of labels and identifying any imbalances.
- **Confusion Matrix**: A heatmap showing the true labels versus the predicted labels. It helps in understanding which classes are being confused with each other.

A.2 Approaches Solving CLuster Loss Problem

A.2.1 Regularization Techniques

a L2 regularization was employed to prevent overfitting by adding a penalty to the loss function proportional to the sum of the squared values of the model parameters. This technique helps in discouraging the model from learning overly complex or large weights.

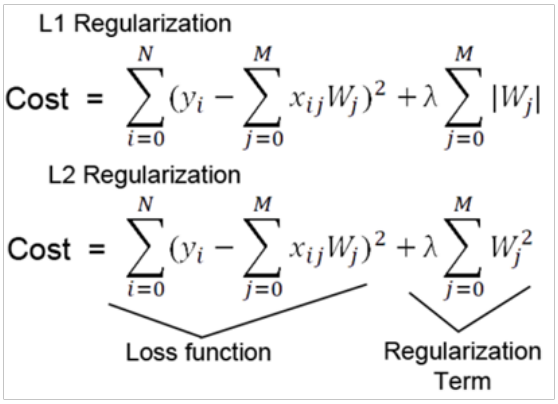


Figure A.1: L1- and L2-regularization

A.2.2 Warm-Up Stage for Cluster Loss

A warm-up stage was used for the cluster loss to gradually increase its influence over the initial training steps, stabilizing the training process. This method aims to allow the model to learn more robust features before applying the full cluster loss.

Listing A.1: Warm-Up Stage for Cluster Loss

```
self.warmup_steps = 1000

if self.global_step < self.warmup_steps:
    warmup_factor = self.global_step / self.warmup_steps
else:
    warmup_factor = 1.0

cluster_loss, cluster_probs = self.cluster_probe(detached_code, None)
cluster_loss *= warmup_factor
```

A.2.3 Cluster Initialization Methods

Different initialization strategies for the cluster centroids were tested, including Xavier and uniform initialization. These strategies were implemented to explore different initial conditions for the clusters, aiming for better convergence.

Listing A.2: Cluster Initialization Methods

```
if self.init_method == 'xavier':
    xavier_init = torch.empty(self.n_classes, self.dim)
    torch.nn.init.xavier_uniform_(xavier_init)
    self.clusters.copy_(xavier_init)
elif self.init_method == 'uniform':
    uniform_init = torch.rand(self.n_classes, self.dim).uniform_(-1, 1)
    self.clusters.copy_(uniform_init)
```

A.2.4 Positive Offset in Cluster Loss

A positive offset was added to the cluster loss to ensure that the initial loss values were not too small, promoting better gradient flow during the initial training phase. This approach aims to prevent the cluster loss from starting too low and not contributing effectively to the optimization process.

Listing A.3: Positive Offset in Cluster Loss

```
cluster_loss, cluster_probs = self.cluster_probe(detached_code, None)
cluster_loss += 1e-3  # Positive offset
```

A.2.5 Entropy-Based Loss Function

A problem issued on the STEGO repo page was about the cluster loss calculation and how it should actually follow the entropy loss function. See Figure A.2. An entropy-based loss function is used to encourage a more uniform distribution of cluster assignments, preventing collapse to a few clusters. This loss function aims to distribute the assignments more evenly across clusters, improving clustering quality.

Listing A.4: Entropy-Based Loss Function

```
def entropy_loss(cluster_probs):
    return -torch.mean(torch.sum(cluster_probs * torch.log(cluster_probs

cluster_loss, cluster_probs = self.cluster_probe(detached_code, None)
cluster_entropy_loss = entropy_loss(cluster_probs)
loss += cluster_entropy_loss
```

Figure A.2: Entropy Loss problem issued by user

A.3 Competition and Contribution to this work

```
LBC
|-- Privileged Agent (Teacher)
|   |-- Access to BEV semantic segmentation images
|   |-- Learns optimal actions
|   |-- Predicts waypoints
|
|-- Sensorimotor Agent (Student)
    |-- Access to forward-facing RGB images
    |-- Learns to imitate the teacher
    |-- Operates without privileged BEV information

CBS
|-- Adaptation of LBC
|   |-- Uses forward-facing camera images instead of BEV images
|   |-- Practical for real-world scenarios
|
|-- Teacher-Student Architecture
|   |-- Teacher with BEV semantic segmentation
|   |-- Student with forward-facing RGB images
|
|-- Performance
    |-- Maintains LBC advantages with practical sensory inputs
```

```
CBS2
|-- Extension of CBS
|   |-- Optimized data collection parameters
|   |   |-- Number of frames
|   |   |-- Steering noise
|   |   |-- Traffic light detection distance
|   |-- Advanced perception modules
|       |-- Pyramid Pooling Module (PPM)
|       |-- Feature Pyramid Network (FPN)
|
|-- Teacher-Student Architecture
|   |-- Teacher with BEV semantic segmentation
|   |-- Student with forward-facing RGB images
|
|-- Performance
    |-- Improved lane-keeping and reduced infractions

WoR
|-- Model-Based Reinforcement Learning
|   |-- Ego-Vehicle Model
|   |   |-- Predicts next state based on current state and action
|   |   |-- Simulates vehicle behavior without environmental interaction
|   |-- World Model
|       |-- Pre-determined environment states
|       |-- Independent of the agent's actions
|
|-- Q-Values
|   |-- Computed using predefined reward function
|   |-- Rewards lane-keeping
|   |-- Penalizes collisions and traffic infractions
|
|-- Performance
    |-- Higher driving score on CARLA leaderboard
    |-- Less data usage
    |-- Excellent performance on NoCrash benchmark

Your Thesis
|-- Builds on LBC
```

```
|   |-- Uses teacher-student architecture
|   |-- Introduces semantic segmentation to the student agent
|
|-- Advances CBS and CBS2
|   |-- Optimized data collection
|   |-- Advanced perception techniques
|   |-- Real-time segmentation
|
|-- Incorporates WoR's Efficiency
    |-- Model-based insights
    |-- Efficient data handling and training
```

A.4 Real-Time Segmentation Code

```
import cv2
import torch
import torch.nn.functional as F
import numpy as np
import threading
from queue import Queue
from train_segmentation import LitUnsupervisedSegmenter
from utils import get_transform
from crf import dense_crf
from os.path import join
from PIL import Image

# Paths and model loading
dir = "path/to/model/directory/"
sav_model = "saved_model.ckpt"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load the PyTorch model
model = LitUnsupervisedSegmenter.load_from_checkpoint(join(dir, sav_mode

vid_path = "../../testing_videos/"
video = "vid_town10HD_small_crash.mp4"
video_path = join(vid_path, video)

# Frame queue for real-time processing
frame_queue = Queue(maxsize=100)
```

```
segmented_frame_queue = Queue(maxsize=100)

# Batch size for processing frames
BATCH_SIZE = 4
resize_res = 448

def process_batch_with_stego(frames, model, use_linear_probe=True):
    original_height, original_width = frames[0].shape[:2]

    # Resize and transform frames for segmentation
    resized_frames = [cv2.resize(frame, (resize_res, resize_res), interp
    transform = get_transform(resize_res, False, "center")
    imgs = torch.stack([transform(Image.fromarray(resized_frame)).to(dev

    with torch.no_grad():
        codes = model(imgs)
        codes_flipped = model(imgs.flip(dims=[3]))
        codes = (codes + codes_flipped.flip(dims=[3])) / 2
        codes = F.interpolate(codes, imgs.shape[-2:], mode='bilinear', a

        segmented_frames = []
        for img, code in zip(imgs, codes):
            if use_linear_probe:
                linear_probs = torch.log_softmax(model.linear_probe(code
                seg_pred = dense_crf(img.cpu().float(), linear_probs[0].
            else:
                cluster_probs = model.cluster_probe(code.unsqueeze(0), 2
                seg_pred = dense_crf(img.cpu().float(), cluster_probs[0]

            segmented_frame = model.label_cmap[seg_pred].astype(np.uint8
            segmented_frame = cv2.cvtColor(segmented_frame, cv2.COLOR_BG
            segmented_frame = cv2.resize(segmented_frame, (original_widt
            segmented_frames.append(segmented_frame)

    return segmented_frames

def read_frames():
    cap = cv2.VideoCapture(video_path)
    while True:
        ret, frame = cap.read()
```

```
            if not ret:
                frame_queue.put(None)
                break
            frame_queue.put(frame)
        cap.release()

    def process_frames():
        while True:
            batch_frames = []
            while len(batch_frames) < BATCH_SIZE:
                frame = frame_queue.get()
                if frame is None:
                    break
                batch_frames.append(frame)
            if not batch_frames:
                segmented_frame_queue.put(None)
                break
            segmented_frames = process_batch_with_stego(batch_frames, model)
            for original_frame, segmented_frame in zip(batch_frames, segment
                segmented_frame_queue.put((original_frame, segmented_frame))

    def display_segmented_frames():
        while True:
            frame_pair = segmented_frame_queue.get()
            if frame_pair is None:
                break
            original_frame, segmented_frame = frame_pair
            combined_frame = np.hstack((original_frame, segmented_frame)).as
            cv2.imshow('Segmented Frame', combined_frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        cv2.destroyAllWindows()

    reader_thread = threading.Thread(target=read_frames)
    processor_thread = threading.Thread(target=process_frames)
    display_thread = threading.Thread(target=display_segmented_frames)

    reader_thread.start()
    processor_thread.start()
    display_thread.start()
```

```
reader_thread.join()
processor_thread.join()
display_thread.join()

print("Real-time video segmentation completed.")
```

A.5 README

This repository contains the implementation of an unsupervised segmentation model specifically designed for autonomous driving applications. The goal of this project is to develop a robust segmentation algorithm that can operate without labeled data, providing a scalable solution for real-world autonomous driving scenarios.

The project consists of multiple key sections in order to perform the semantic segmentation. Below you see the general steps outlined to get a rough idea:

1. Data Collection
2. Data Processing To Desired Format
3. Cropping Utility To Improve Spatial Resolution
4. Precomputation of KNN indices
5. Training of Model
6. Evaluation
7. Real-Time Segmentation or Video/Image Segmentation

A.5.1 Code Source

This repository contains code from other sources:

- Modified:
  - Cheating by Segmentation 2 (branch: cbs2)
  - STEGO
- Not modified/implemented:
  - DriveAndSegment

A.5.2 Installing Carla

Install Carla in the desired location:

```
wget https://carla-releases.s3.eu-west-3.amazonaws.com/Linux/CARLA_0.9.10.1.tar.gz
tar -xvzf CARLA_0.9.10.1.tar.gz -C carla09101
```

Clone USAD repository in desired location:

```
# Clone the repository
git clone https://github.com/flixtkc/Unsupervised-Segmentation-for-Autonomous-Drivi

# Navigate into the project directory
cd Unsupervised-Segmentation-for-Autonomous-Driving-Cars
```

For this project, there are two distinct environments you need to create. If not already done, install conda.

```
cd CBS2
conda env create -f docs/cbs2.yml
conda activate cbs2
```

Once tested to see if the environment is set up correctly, continue with the second environment (first go back to the main directory).

```
cd ..
cd STEGO
conda env create -f environment.yml
conda activate stego
```

Add the following environment variables to `~/.bashrc`:

```
export CARLA_ROOT=<your_path>/carla09101
export CBS2_ROOT=<your_path>/CBS2
export LEADERBOARD_ROOT=${CBS2_ROOT}/leaderboard
export SCENARIO_RUNNER_ROOT=${CBS2_ROOT}/scenario_runner
export PYTHONPATH=${PYTHONPATH}:"${CARLA_ROOT}/PythonAPI/carla/":"${SCENARIO_RUNNER
```

Verify the setup by launching Carla (with cbs2 virtual environment activated):

```
source ~/.bashrc
$CBS2_ROOT/scripts/launch_carla.sh 1 2000
```

A.5.3 Setup Configurations

To set up configurations for the data collection, there are several files to consider. Open any text editor of your choice and inspect the following files:

```
CBS2/autoagents/collector_agents/config_data_collection.yaml
CBS2/autoagents/collector_agents/collector.py
CBS2/rails/data_phase1.py
CBS2/assets/
```

config_data_collection.yaml

This file contains all the settings that define the data collection phase. Especially the resolution settings are key in determining how well the segmentation model will perform.

collector.py

This file specifies all the collector functions. It also includes configurations for logging to Weights & Biases (wandb), which is set to False by default.

data_phase1.py

The former denotes all the settings that define the data collection phase. The second one specifies all the collector functions, where also the logging to wandb can be found (default False). The latter contains the settings to set the driving episode-specific config.

assets/ directory

Here you can find the routes and scenarios for each town, which are needed to run any data collection successfully. You need to specify which one to use in data_phase1.py.

A.5.4 Data Collection

Make sure to start Carla in one terminal, then boot up a second screen/terminal, go to the CBS2 subdirectory there, and run the data collection script:

```
# Terminal 1:
$CBS2_ROOT/scripts/launch_carla.sh <num_runners> <port>

# Terminal 2:
```

```
cd CBS2
python rails/data_phase1.py --port <port> --num-runner=<num_runners>
```

Note: if there are multiple runners, `<port>` is also the increment between them.

A.5.5 Data Processing To Desired Format

After you have verified the collected data and are content with the results, you can continue to the data conversion step. Where the saved data has to be processed before the STEGO model can effectively train with it. Go back to the main directory where you can find the lmdb_to_STEGO_dataset converter script. Inspect the possible arguments passed to the functions before running this script. Note: the dataset input path and output path need to be specified.

Switch Environments

After this step, it is important to deactivate cbs2 and activate the stego environment, as the next steps will need that adjustment.

A.5.6 Configurations for STEGO

Inspect the configuration for the STEGO related script in:

```
cd STEGO/src/
vi configs/train_config.yml
```

Specify the dataset path, which is identical to the output path used in the previous step. For the next step, you need to adjust the hyperparameters related to the cropping as these heavily influence memory-related errors.

A.5.7 Cropping Utility

```
has_labels: False
crop_type: "five"
crop_ratio: .5
res: 200
loader_crop_type: "center"
```

The `crop_type` determines whether it takes 5 crops one from each corner and then one in the middle, or, random crops from the input image(s). Now the `crop_ratio` determines how big/small the resulting crops will be compared to the input dimensions of the image. Tweak until you have favorable settings. After cropping, please adjust the dataset path in the config file to point to the newly created cropped dataset directory.

A.5.8 Precomputation KNN Indices

To speed up training steps, it is crucial to run a precomputation of KNN indices:

```
python precompute_knns.py
```

A.5.9 Train STEGO Model

Now all that is left is to run a training script on your custom data to see the segmentation results of the STEGO model on the collected Carla simulator data. Please specify in the config file the logs directory:

```
python train_segmentation.py
```

During training, you can monitor the entire phase by running a tensorboard session in the specified logs directory to see some intermittent progress:

```
# Example
tensorboard --logdir logs/logs/five_crop_0.5/directory_new_crop_date_Jul25_02-25-32
```

A.5.10 Evaluation and Testing

To evaluate and compare your training results, you can monitor tensorboard logs of course where you can see various metrics over time, but you can also run a real-time segmentation script to see a real-life application. As discussed in the thesis, this resulted in no real-time application but will be improved upon in the future. Additionally, there is also a script that takes a normal video and segments it for you given the specified model, and then saves the video. You can uncomment the image or video segmenter, whichever one you prefer. For an example please see the `testing_videos/` directory!

```
cd STEGO/src/
python STEGO_create_segmented_video_or_image.py
python STEGO_real_time_segmenter.py
# See
cd ../../testing_videos/
```

A.5.11 Extra Tests

Additionally, there are several scripts to test your system's batch size, number of workers, and ssh X11 forwarding in case you run into errors.

A.6 Augmentation Bloopers

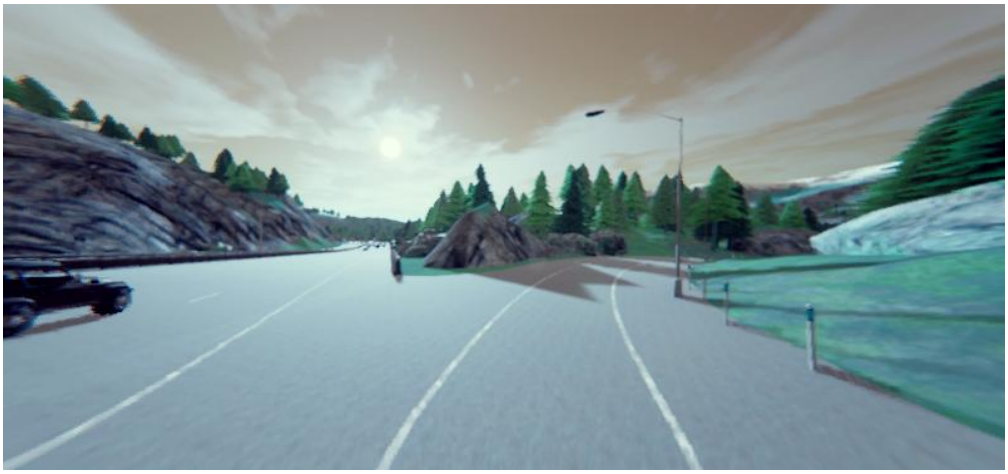


Figure A.3: Blooper 1

Figure A.4: Blooper 2

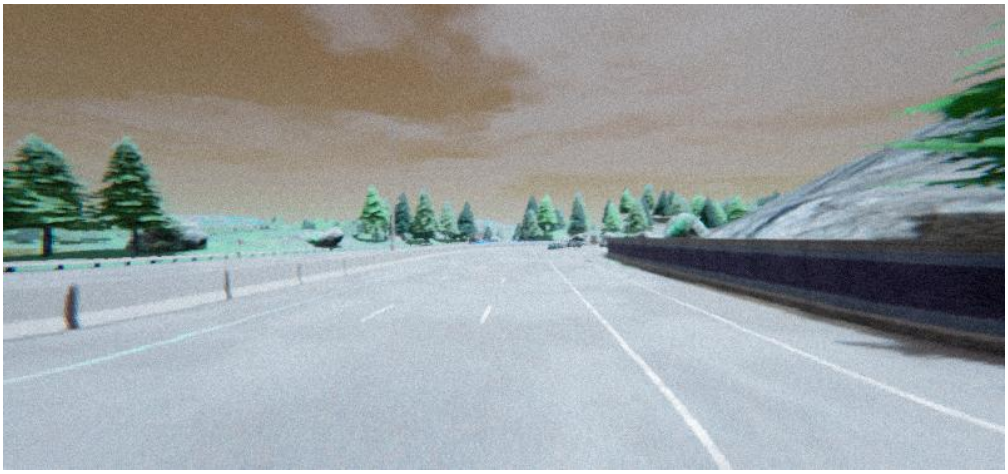


Figure A.5: Blooper 3

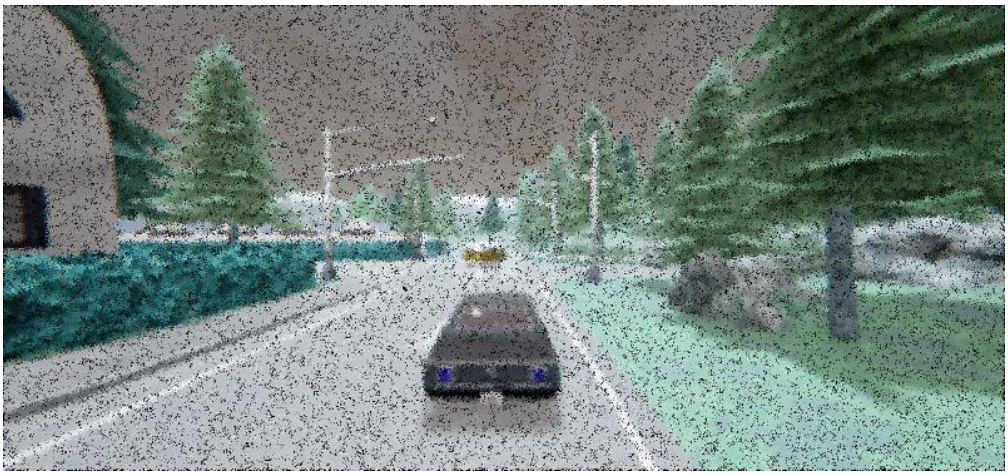


Figure A.6: Blooper 4

Figure A.7: Blooper 5

Figure A.8: Blooper 6

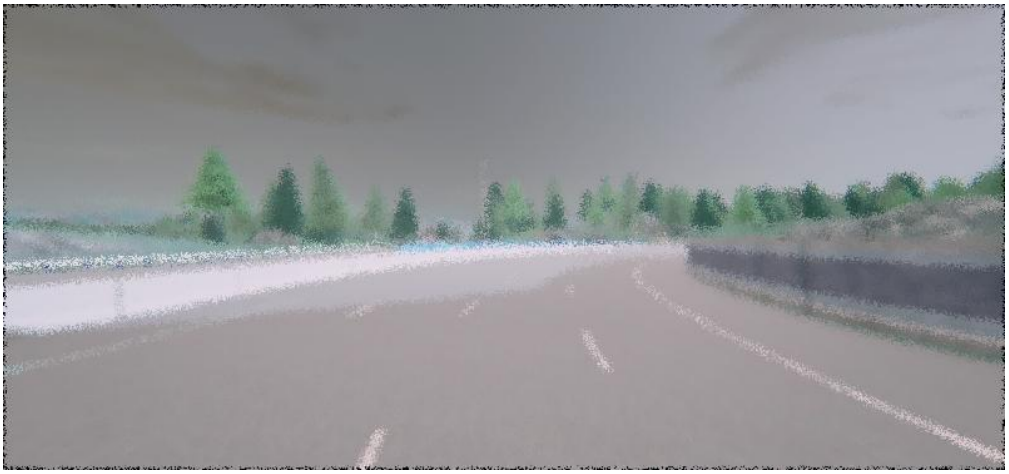


Figure A.9: Blooper 7

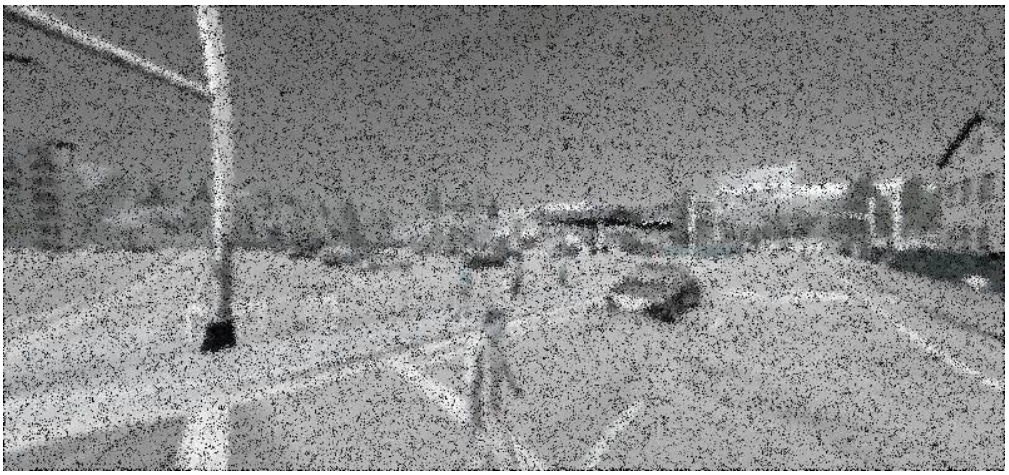


Figure A.10: Blooper 8