UPPSALA
UNIVERSITET

Instance embedding for clustering headlights

Emilija Maikstenaite, Ganesh Sudarshan, Jacob Henningsson, Yufeng Chen

February 27, 2024

1 Introduction

Autonomous vehicles are at the forefront of technological advancements, aiming to replicate and even surpass human drivers. One of the significant challenges these vehicles face is nighttime driving, where the absence of light poses unique detection and navigation challenges. Humans rely on visual cues such as headlights to identify vehicles during the night, it's natural to have solutions emulate this intuitive process. The objective of this project is to implement a machine-learning model that not only detects headlights but also clusters them into individual vehicles. These clusters can then be used to determine the vehicle's positions. The project owners have specified that an embedding space should be used for clustering the headlights in combination with a push-and-pull loss function.

2 Background

2.1 Related work

The field of autonomous driving covers a lot of different technologies. The focus of the presented related work will be specifically on the detection of headlights. Blob detection is a common headlight detection technique that can be divided into two stages: blob segmentation and blob classification [1]. Blob detection has two main approaches: auto-exposure, which segments bright elements in the image, and low-exposure, where thresholding is used across the entire image to produce blob segments [2]. Once detected, a machine-learning method is utilized to classify the blobs. Blob detection is not meant for the accurate positioning of headlights and is mostly used to control automated high-beam systems, where accurate positioning is not a requirement.

Ligayo et al. proposed a single-step solution using the YOLOv3 architecture [3]. YOLO (You Only Look Once) is a CNN-based architecture used for object detection. Using this architecture the authors were able to achieve an accuracy of 86.72% trained on a dataset consisting of 3000 images. It should be noted that this strictly focused on headlight detection and not on instance clustering per vehicle.

Before a vehicle comes into view, human drivers rely on the reflected light from headlights to determine if there is an oncoming vehicle. This scenario can occur when oncoming vehicles are approaching from around a corner or over a hill. This method is referred to as providence or foresight setting it apart from the more usual detection scenarios where the object is fully visible. The objective is to identify an object not based on its direct position in the image but by utilizing attributes that provide clues to its location outside of the frame. Ewecker et al. propose a custom detection pipeline to detect oncoming vehicles by utilizing provident attributes [4]. The main components of the pipeline include object detection, distance estimation, and object tracking. The implementation of this pipeline enabled the authors to detect vehicles 1.7 seconds faster than solutions that did not integrate provident attributes.

2.2 Associative Embedding

Newell et al. present a solution that outputs both detection and groupings at the same time [5]. The authors used this solution for pose estimation and instance segmentation of images containing multiple people. Performing both detection and grouping at the same time is beneficial since the two are highly correlated. The solution uses the stacked hourglass architecture which was developed for dense pixel-wise prediction [6]. The architecture consists of a "stacked" sequence of "hourglass" modules which can be seen in figure 1. An example of one of these modules can be seen in Figure2, each box in the figure corresponds to a residual module. Each

"hourglass" has a standard set of convolutional and pooling layers that down-samples features to a low resolution capturing the full context of the image. Then, these features are gradually up-sampled and combined with residuals from the down-sampling operations until they reach the final output resolution. Stacking multiple hourglass modules enables repeated bottom-up and top-down inference to produce a more accurate final prediction[5]. To get the final prediction the output from all of the hourglass modules is combined by either taking the mean or sum of the outputs.

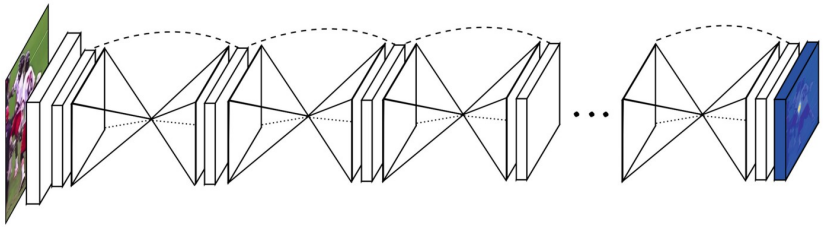


Figure 1: Stacked hourglass model: consists of multiple stacked hourglass modules which allow for repeated bottom-up, top-down inference[6].

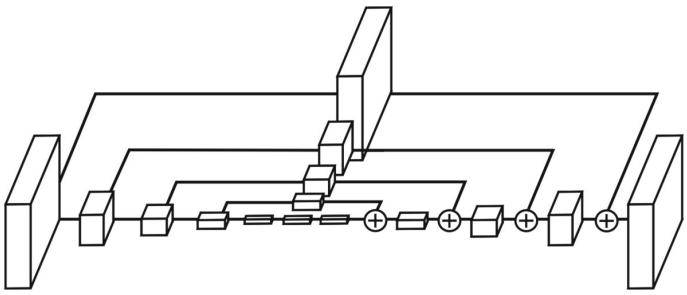


Figure 2: An illustration of a single "hourglass" module.[6].

The stacked hourglass model, initially designed for single-person human pose estimation, was modified in [6] to enable multi-person pose estimation and instance segmentation.

For single-person pose estimation, the model generates a detection heatmap for each body joint. Subsequently, the location of each joint is predicted by identifying the pixel with the highest activation within the corresponding heatmap. The architecture is designed to combine information from global and local features, thereby encompassing information about the overall body structure while retaining intricate joint details for accurate detection. This consideration of both global and local features proves to be equally crucial in various other pixel-wise prediction tasks such as segmentation [5]

For multi-person pose estimation, an optimal detection heatmap should exhibit several peaks, facilitating the identification of multiple joints, such as multiple left wrists belonging to different people. This creates the additional task of grouping different categories of joints to construct one person. To facilitate this the model outputs a "tag"-map in addition to the heatmap for each joint. If there are m body joints to predict then the model will output a total of 2m maps, m number for detection and m number for grouping. The resolution of the tagmap matches that of the heatmap, each pixel in the tagmap is assigned a value to represent its instance. The tag values are used to group each joint category into one person. This essentially means that the "embedding space" of the grouping is 1D since they are represented by a single value in the tagmap. An example of the output of the model can be seen in figure 3.

To parse the output into a joint with a tag value, non-maximum suppression is used to get the peak values in each heatmap. Using the coordinates of the peak heatmap values the corresponding tag values are extracted from the tag map. Then detections are clustered across body parts by finding the tag values that are the closest across different body joints. A group of detections now forms the pose estimate for a single person[5].

2.3 PVDN dataset

The Provident Vehicle Detection at Night (PVDN) dataset comprises images captured during a drive at night where the headlights of oncoming traffic are annotated [7]. In addition to the "direct" headlight annotations, "indirect" headlight annotations show light reflected from headlights in the images. The authors state that this information is useful since drivers can use the reflections as "foresight" that an oncoming vehicle will appear and for instance disengage the high beams. These "provident" reflections are unique to computer vision solutions

Figure 3: Example of an output from the model showing the input image, detection map, and tag map [5]

since techniques such as LIDAR can't detect objects without line-of-sight. Images were additionally captured during the day using a short exposure to give them the appearance of being captured at night, the differences between day and night images can be seen in figure 4.

(a) Example of an image captured at night using a long exposure.

(b) Example of an image captured during the day using a short exposure.

Figure 4: Difference between images captured during the day and night

2.3.1 Dataset structure

The dataset consists of grayscale images captured by a camera onboard a car. Images are frames extracted from video sequences that start right before a vehicle approaches and end right after the vehicle has passed. Each sequence of images is placed in separate folders in the dataset.

```
{"image_id": 19294,
"annotations": [
    {"oid": 0, "pos": [264, 671], "instances": [
            {"iid": 0, "pos": [192, 618], "direct": true, "rear": false},
            {"iid": 1, "pos": [364, 615], "direct": true, "rear": false},
            {"iid": 2, "pos": [196, 800], "direct": false, "rear": false},
            {"iid": 3, "pos": [367, 767], "direct": false, "rear": false}],
        "direct": true},
    {"oid": 1, "pos": [843, 492], "instances": [
            {"iid": 0, "pos": [1177, 424], "direct": false, "rear": false}],
        "direct": false}],
"blurs": [[229, 632, 305, 660]]}
```

Figure 5: Annotation structure example from the PVDN dataset [8].

Each image annotation can contain multiple vehicle annotations with the "oid" identifier seen in figure 5. The vehicles can contain multiple headlight "instances" identified by "iid". These sources of light can have two attributes "direct" and "rear". "direct" is intended to differentiate between reflected light and light that comes directly from a headlight, while "rear" is used for taillights. The vehicles have coordinates that mark their position "pos". When the vehicle has only indirect light instances, the position is estimated to be where the vehicle is anticipated to appear. For vehicles where both headlights are "direct", the position of the vehicle is placed between them on the surface of the road.

(a) Vehicle indirect position

(b) Vehicle direct position

Figure 6: Examples of vehicle annotated positions

3 User requirements

The purpose of this project is to evaluate push-and-pull loss functions and architectures for the project owner. The architecture and loss functions are specified beforehand by the project owner. The goal is not necessarily to achieve the highest accuracy but to implement the specified solutions.

The solution shall detect and group vehicles by their headlights
The model shall input an image and output detections along with information on how to cluster them. Specifically, the model shall solve the task of headlight detection and per-vehicle clustering in a single night-time image. For training the PVDN (Provident Vehicle Detection at Night) dataset [7] shall be used.

The model shall use a push-and-pull loss function.
The model shall be trained using a push-and-pull loss function. This will help the user evaluate the viability of using such a loss function. This also entails that the model implements an architecture that can take advantage of a push-and-pull approach.

The solution shall visualize predictions.
A user shall be able to visualize predictions on a provided image or image sequence. The visualization shall be in the form of the original image overlayed with detected keypoints where the clusters are made distinct using a different color per vehicle. This will help the user visually evaluate the architecture/loss-function approach. The visualizations shall also include raw model output for debugging purposes.

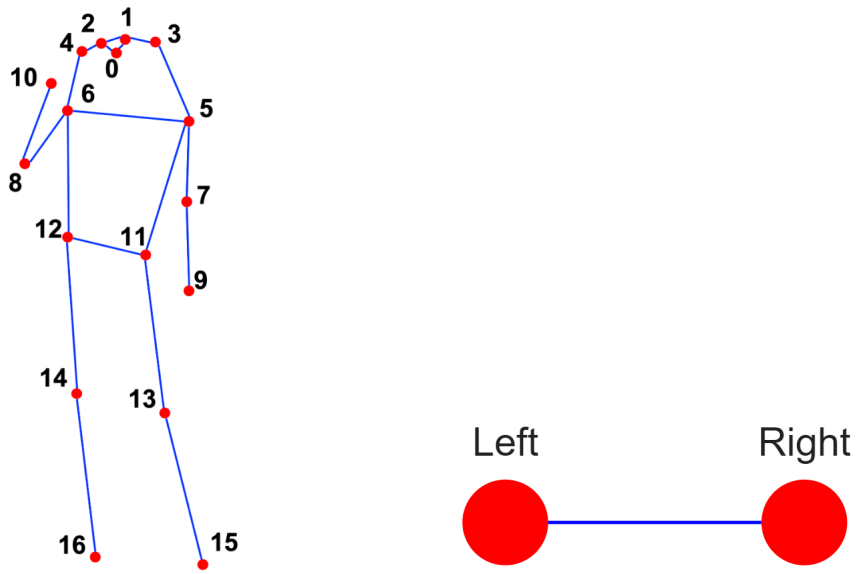
The model shall be evaluated on the PVDN test set
The implemented evaluation method shall use the PVDN test set to quantify the performance of the model.

The model shall output an embedding space for grouping
The embedding space can be used to visualize the instance clustering of the vehicle, this can further evaluate the performance of the model.

4 Your solution

4.1 Network Architecture

The architecture and loss function in [5] was used as a basis for this project. Modifications were made to solve the task of headlight detection at night. The model outputs m*2 channels where m is the number of classes, in the original paper they used 17 classes for each joint of the body. For this project, we use 2 classes: right headlight and left headlight, that are then clustered to make one vehicle. This clustered pair can now be used to find a vehicle's position. A comparison between the person skeleton and our vehicle skeleton can be seen in figure 7. This means that we changed the output dimension of the model from 34 to 4 dimensions.



(a) Person skeleton

(b) Vehicle skeleton

Figure 7: Showing the difference between the skeleton used in the reference paper and our solution

As mentioned previously the PVDN dataset comprises grayscale (1 dimension) images, the original architecture was made for color RGB images (3 dimensions). The architecture was thus changed to input 1 dimension which reduced the size of the final model.

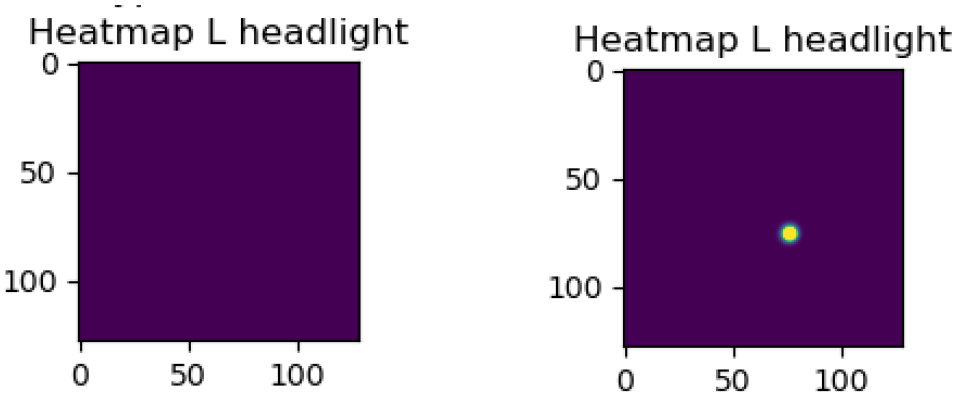
4.2 Model output parsing

The model outputs a detection map and a tag map per classification. To find the detected position of the headlight and its clustering this output needs to be post-processed. The position needs to be found in the detection map and the corresponding tag value needs to be found in the tag map.

4.2.1 Detection map thresholding

The detection maps contain "scores" for how likely each pixel is to contain the target class. To find the coordinates of the most likely headlight detections two thresholding steps are used. Two steps are used since it's difficult to find just one hard threshold that works in all instances. This difficulty comes from that the range from the lowest to the highest value in the detection maps is different between predictions. Setting the value too high excludes detections from some predictions and setting it too low may produce too many detections.

When there is no vehicle in the image the values are generally low which can be seen in the predicted heatmap in 8a. Thus we set a low initial threshold to remove all values that couldn't possibly be a headlight (about 0.5). The next step is to find the highest of the remaining values and set it as the new thresholding value minus 5% of its value. This ensures that the thresholding is set to an appropriate value for headlight detection.



(a) Predicted heatmap when there is no vehicle in the image

(b) Predicted heatmap when there is a vehicle in the image

Figure 8: Difference between heatmap with and without a vehicle in the image

4.2.2 Tag map clustering

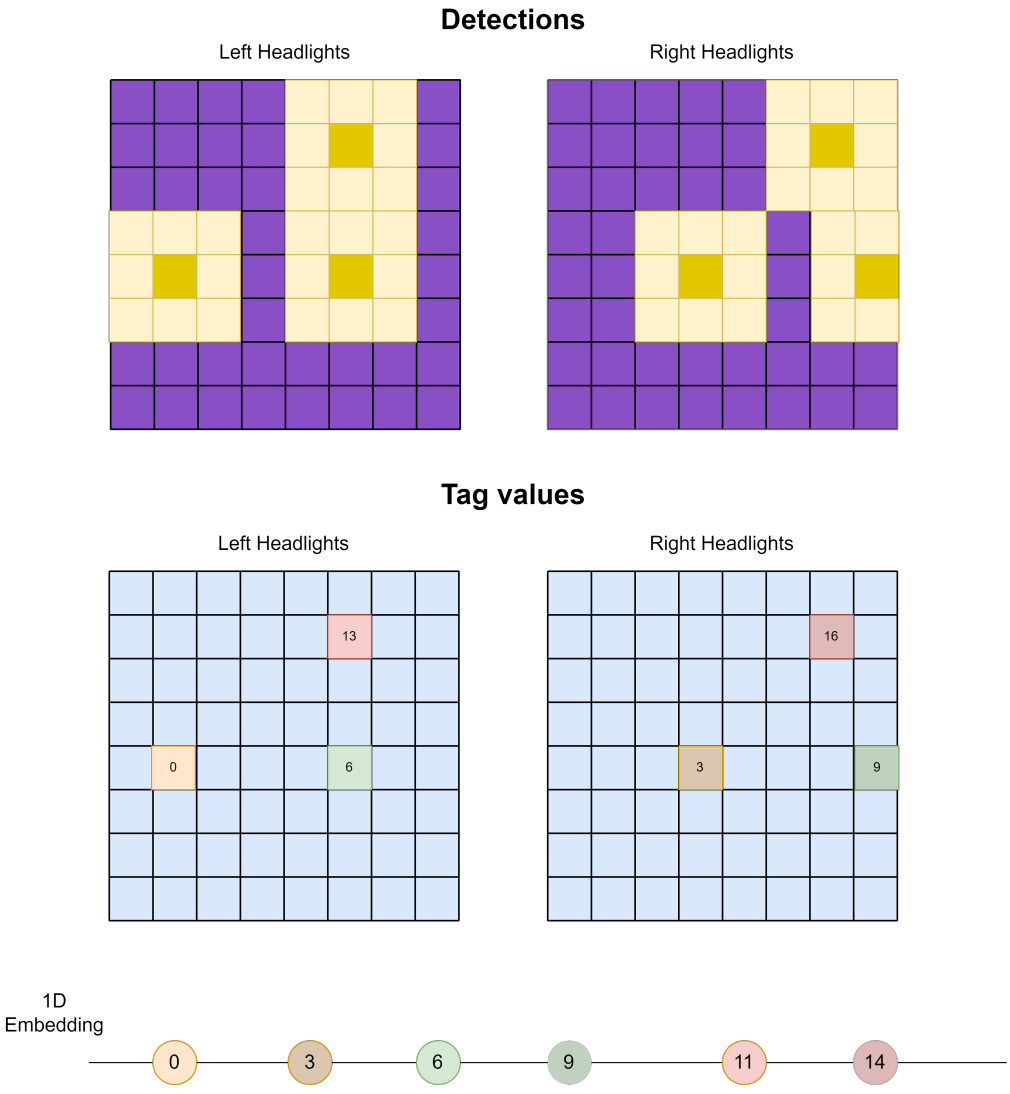


Figure 9: Enter Caption

The tag maps contain the corresponding tag value of each pixel in the detection map. After finding the location of the headlights in the detection map the tag values are found by checking the same location in the tag map. Once each detected coordinate has its associated tag value then it's a matter of finding which pairs have the

closest tag values. A simplified heatmap and tagmap scenario can be seen in figure 9. The challenges when pairing the headlights are:

- There can be an unequal number of left and right headlights.
- The total distance between all headlights needs to be maximized not just the distance in each pair.
- The solution needs to be relatively quick.

We introduce a method that satisfies each of these. Since this project uses a 1D embedding space for the tag values they can be represented on a line, in 10 an example of this can be seen. Clustering greedily from using the left headlights we get the clustering result seen in figure 11, this does not only find a sub-optimal clustering but also incorrectly identifies the outlier. The greedy approach starts with the first left headlight and selects the right headlight that is the closest to it without regard if it's the closest to that particular right headlight.

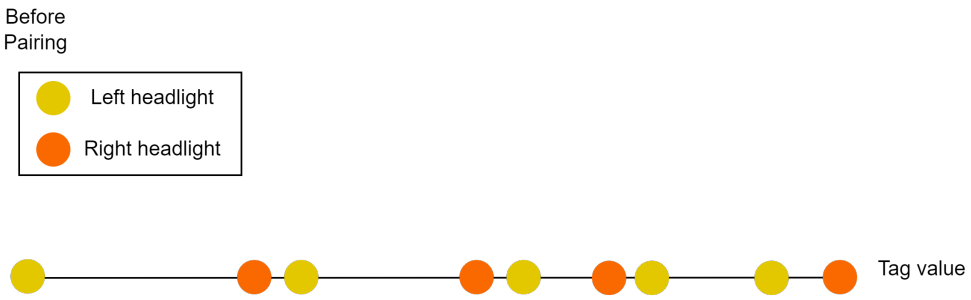


Figure 10: Example of embedding space with left and right headlight to cluster into pairs

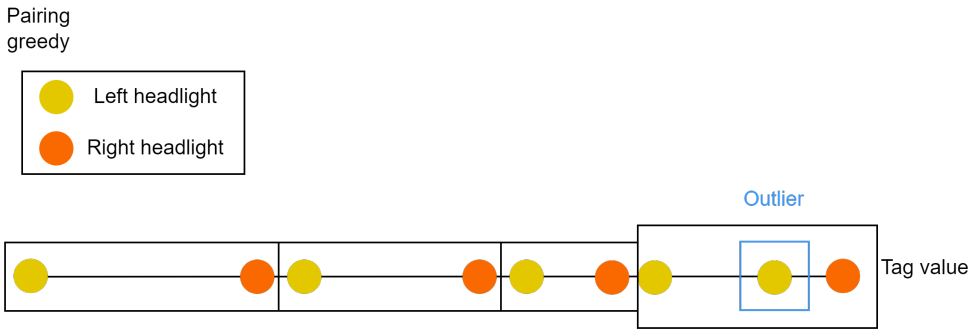


Figure 11: Example using greedy pairing

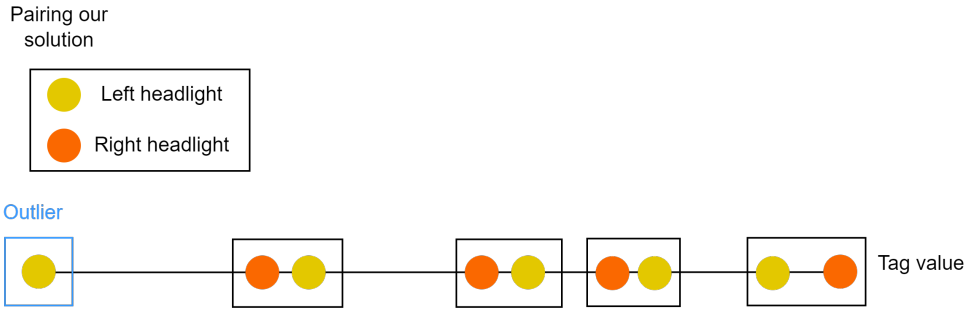


Figure 12: Example using our solution

Our approach considers the closest distance of both the headlights in a pair and can correctly identify the outlier in this example which can be seen in 12. To organize data in the algorithm a headlight class is used, the structure of this class can be seen in listing 1. The attributes in the class are headlight coordinates

(coordinates), headlight tag value (tag_value), headlight of the other class that this headlight is clustered with (pair), distance to the clustered headlight (distance_to_pair), and a list of distances to all headlights of the other class. The algorithm starts by finding the class that has the most headlights and calculates the distance of each of those tag values to the tag values in the other class. The result will be a list of distances that is stored in the headlight class attribute "distance_to_pair". The next step is to iterate over the class containing the most headlights and find the closest unpaired headlight of the other class. If the closest light is already paired check if the current headlight is closer than the headlight that it's already paired with, then pair it with the current headlight instead. This iteration continues until all headlights have found their closest pair. At the end of the algorithm, there could exist several outliers from the more numerous classes. Pseudo code for this algorithm can be seen in 19.

Listing 1: Pseudocode for Headlight class

```
class Headlight:
    # Data
    coordinates: Tuple
    tag_value: Float
    pair: Headlight
    distance_to_pair: Float
    distances: List
```

Algorithm 1: Headlight clustering algorithm

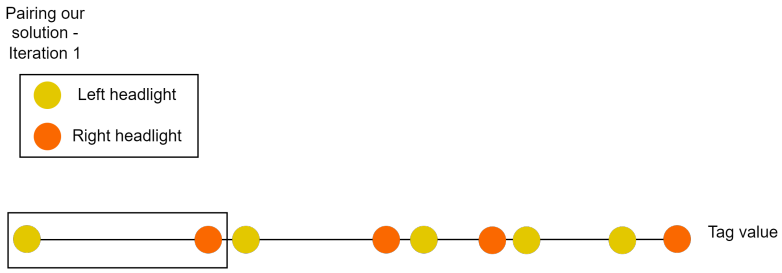
Data: Left/right coordinates, left/right tag values
Result: List of clusters coordinates

Find the class that has the most headlights (If they are the same it doesn't matter)
headlights_L = max(right_headlights, left_headlights)
headlights_S = min(left_headlights, right_headlights)

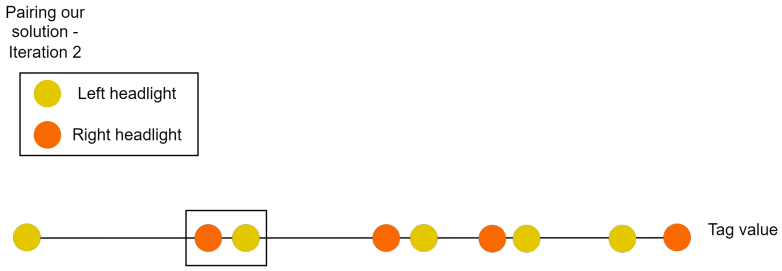
forall *Headlights in headlights_L* **do**
    Calculate the distance to each headlight in the headlights_S and store it in the distances list of the headlight from the longer list.

while *True* **do**
    Continue_loop = False **forall** *headlight_L in headlights_L* **do**
        forall *idx, distance in enumerate(headlight_L.distnaces)* **do**
            if *headlight_L has no pair AND headlights_S[idx].distance_to_pair < distance* **then**
                headlight_L.pair = headlights_S[idx].distance
                Continue_loop = True
    if *Not Continue_loop* **then**
        Break

forall *headlight_L in headlights_L* **do**
    if *headlight_L.pair* **then**
        Cluster [headlight_L.coordinates, headlight_L.pair.coordinates]
    else
        Cluster [headlight_L.coordinates, None]



(a) First iteration of our solution.



(b) Second iteration of our solution.

Figure 13: Two iterations of our solution.

To clarify how the algorithm works the first two iterations of the algorithm can be seen in figure 13. Starting with the first left headlight the closest right headlight is the first right headlight thus the pairing seen in figure 13a is made. In the next iteration, the second left headlight is also closest to the first right headlight. After comparing its distance to the distance between the existing paired headlights it replaces the other left headlight since its distance to this right headlight is shorter. The first left headlight is now unpaired again and will check for a new right headlight (Which it will not find in the end).

4.3 Data Loader

The data loader prepares images and annotations to train and evaluate the model. The loader has several support functions to prepare the data based on the chosen configuration. During preprocessing, the data loader filters key points, resizes images, and generates heatmaps. There are also options for calculating class weights, using samples from the day dataset and only reading data from one sequence folder.

4.3.1 Preprocessing

The PVDN dataset contains grayscale images captured at night with accompanying annotations in the form of coordinates. The first step in preprocessing is to find the coordinates that annotate direct headlights. As discussed previously there are other types of light annotated in the dataset that have to be filtered out. This is done by only extracting coordinates that have the attributes "direct" = True and "rear" = False, meaning that the light is not a reflection and coming from the front of the vehicle (headlight as opposed to taillight). After extraction of the appropriate headlight belonging to an image both are scaled to match the model output (512x512).

One of the model outputs is a heatmap giving per-pixel headlight detection scores. Ground truth heatmaps which are generated from the extracted coordinates are required to train the detection part of the model. To generate the heatmap a Gaussian distribution is placed at each coordinate headlight with a fixed sigma for both the left and right headlights.

4.3.2 Weighted dataset

When looking at the PVDN dataset one thing becomes clear, there is an imbalance in the number of images containing different amounts of vehicle instances. Of the images 43% contain no vehicles, 42% contain one vehicle and only 15% contain multiple vehicles. Having such a low number of images where there exist multiple vehicles creates a problem since the purpose of this project is to train a model that can separate vehicle instances. A "WeightedRandomSampler" was implemented using pre-calculated weights to solve this issue. Each image was assigned a weight based on whether it contained zero, one, or multiple vehicles. The goal was to train the model on equal-part images where there are zero, one, or multiple vehicles. The final result will

be that the model sees fewer images containing zero or one vehicle during training and more images containing multiple vehicles. This could however lead to the model becoming overfit on those images containing multiple vehicles since it will see the same images multiple times during one epoch.

(a) Example of an image containing no vehicle.

(b) Example of an image containing one vehicle.

(c) Example of an image containing multiple vehicles.

Figure 14: Examples of the three different image types no vehicle, one vehicle, and multiple vehicles.

4.3.3 Day dataset

This project primarily uses the night portion of the PVDN dataset, but there is a day portion that is captured using a fast shutter speed to give it the appearance of being captured at night. As previously mentioned there exists a lack of images containing multiple vehicles, the choice was made to extract images that contain multiple vehicles from the day dataset and add them to the training data. This will help reduce the risk of overfitting on the small number of images in the night dataset that contain multiple vehicles. This changes the distribution in the dataset to 39% no, 39% one, and 22% multiple vehicles.

4.3.4 Dataset caching

When the dataloader is initiated it needs to create three different dictionaries for annotation paths, annotation/image pairings, and weights. These calculations are done for both night/day and their train/test and validation partitions. When these dictionaries have been calculated they are saved to be used as a cache the next time the dataloder is initiated.

4.4 Model training

4.4.1 Loss function

To train an n-stacked hourglass network, we apply both a detection loss for the heatmaps and a grouping loss for the tagmaps.

The detection loss calculates the mean square error between each predicted detection heatmap and its corresponding ground truth heatmap. This loss function mirrors the one utilized by Newell et al.[6].

The grouping loss measures the alignment between predicted tags and the actual clustering in the ground truth. In simpler terms, it evaluates the model's ability to cluster lights belonging to the same vehicle. This is achieved by the inclusion of both push and pull mechanisms in the loss function. The grouping loss is

designed to minimize distance in the embedding space for all lights in the same vehicle instance (pull force) while simultaneously maximizing the distance between the means of different vehicle instances (push force). Figure 15 illustrates a 1D example of push and pull mechanics.

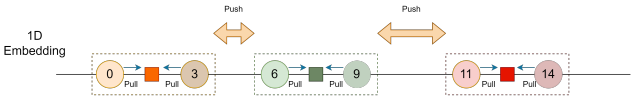


Figure 15: A 1D embedding space example of push and pull mechanics

Formally, let $h_k \in R^{W \times H}(K \in 1, 2)$ be the predicted tagmap for the k-th headlight, where $h(x)$ represents the tag value at pixel location x. Given N cars, let the ground truth light location be $T = (x_{nk}), n = 1, \ldots, N$, $(K \in 1, 2)$, where x_{nk} is the ground truth pixel location of the k-th lights of the n-th car. For example, x_{11} corresponds to the ground truth pixel location of the first car's left light, and x_{12} corresponds to the ground truth pixel location of the first car's right light.

Assuming all K lights are annotated, and the reference embedding for the n-th person would be

$$\overline{h}_n = \frac{1}{K} \sum_k h_n(x_{nk})$$

The grouping loss L_g is then defined as(α and β are the respective weights.)

$$L_g = \alpha \times \frac{1}{N} \sum_n \sum_k (\overline{h}_n - h_n(x_{nk})))^2 + \beta \times \frac{1}{N^2} \sum_n \sum_{n'} exp(\frac{1}{2\sigma^2}(\overline{h}_n - \overline{h}_{n'})^2)$$

And the detection loss L_d is defined as

$$L_d = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{j=1}^{H} \left(h_{ij} - \hat{h}_{ij}\right)^2$$

Therefore, the total loss L during the model training process is given by

$$L = w_1 \times L_d + w_2 \times L_g(h, T)$$

where w_1 and w_2 are the respective loss weights.

Throughout the training process, we configured the aforementioned loss weights as $w_1 : w_2 : \alpha : \beta = 10 : 1 : 1 : 1$.

Figure(16) illustrates the working principle of our CNN model (3-stacked hourglass model) in multi-vehicle light detection and grouping, along with a schematic representation of the loss function computation during model training.

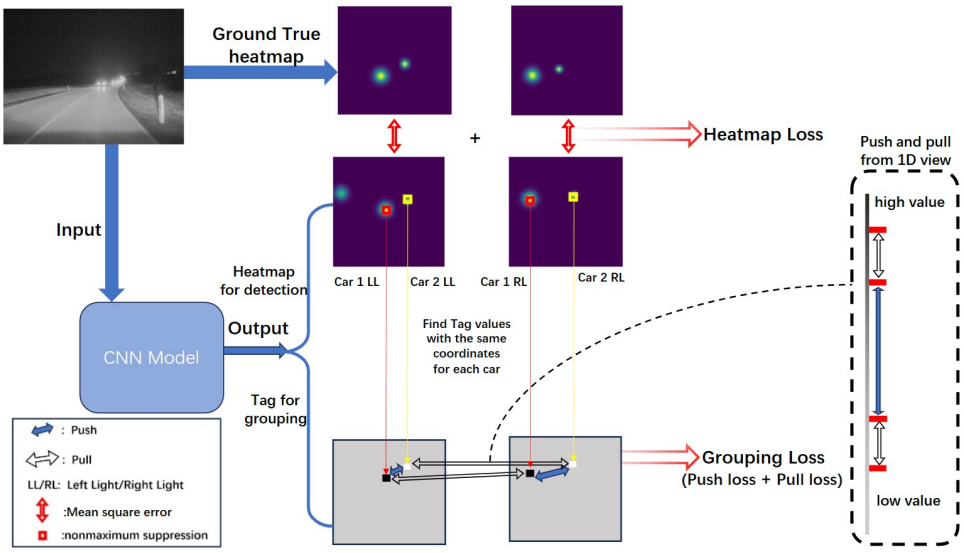


Figure 16: An overview of the working principles and training loss of the stacked Hourglass model. See 4.1 and 4.4 for details.

4.4.2 Hyper parameters

We set the output resolution to 128 * 128 and the input resolution to 512 * 512 images. During the training process, batch normalization was employed with a batch size of 16, and a learning rate of 1e-5, and CosineAnnealingLR was used as a scheduler method to dynamically adjust the learning rate. The loss function ratios for pull, push, and detection were set to 1:1:2.

4.5 Evaluation Metrics

In this project, we utilized the Provident Vehicle Detection at Night (PVDN) dataset for both model training and testing purposes. Notably, in contrast to typical object detection datasets, the PVDN dataset exclusively includes vehicle coordinates and headlight coordinates for the corresponding vehicle for each image. This is in contrast to other object detection datasets that provide bounding boxes. Consequently, conventional evaluation methods, such as the Coco evaluation method for object detection, do not apply to this project since it uses bounding boxes. Therefore, taking into account the dataset, the output characteristics of the model, and the project objectives, we propose a 2D Gaussian evaluation method for the detection and clustering of headlights.

For each input image, the model generates an output that after parsing comprises coordinate pairs corresponding to detected vehicles in the image. Each pair represents the two headlights that are the most likely to be of the same vehicle, in some instances, a vehicle instance will only have one headlight. If the model detects both headlights (left and right), the center point of these headlights (x_i, y_i) serves as the predicted vehicle position. If only one headlight is detected, we assume that the predicted vehicle position is at the headlight's coordinates.

Our method starts by creating a 2D Gaussian F_i for each ground truth vehicle position (X_i, Y_i) in the same 2D array at the corresponding location (X_i, Y_i). The covariance matrix of the 2D Gaussian distribution is

$$Cov = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$$

As σ^2 increases, the 2D Gaussian function becomes flatter; conversely, as σ^2 decreases, the function becomes more peaked.

Following that, the algorithm proceeds to identify the nearest ground truth vehicle position (X_i, Y_i) along with the corresponding 2D Gaussian distribution F_i for each predicted vehicle position (x_i, y_i). The predicted vehicle position (x_i, y_i) is used in the Probability Density Function of F_i, yielding values V_i that will be used in the evaluation. As the predicted vehicle position approaches the true position, the value of V_i approaches 1. Conversely, as the predicted position deviates, V_i decreases (refer to Figure 17).

Ultimately, the average of all predictions V_i is calculated. Additionally, to evaluate the model's ability to detect vehicles and assess the model's over-detection and under-detection of vehicles, while calculating the mean value of V_i, we divide the summation by the larger of the predicted number of vehicles and the true number of vehicles. This ensures that the model's detection score is maximal when the model's predicted number of vehicles aligns with the true value.

The final proposed 2D Gaussian evaluation method can be expressed as:

$$Score = \frac{\sum_i Normalizad(V_i)}{Max(n_t, n_p)}$$

where n_t is the number of ground truth vehicles and n_p is the number of predicted cars.

Furthermore, our code facilitates the generation of evaluation matrices, allowing users to configure multiple σ^2 values simultaneously. By varying σ^2 values, we can observe how the model responds to different degrees of spatial uncertainty. The ability to output evaluation matrices for multiple σ^2 values enhances the versatility of our code, providing users with a nuanced understanding of the model's efficacy in handling variations in spatial information.

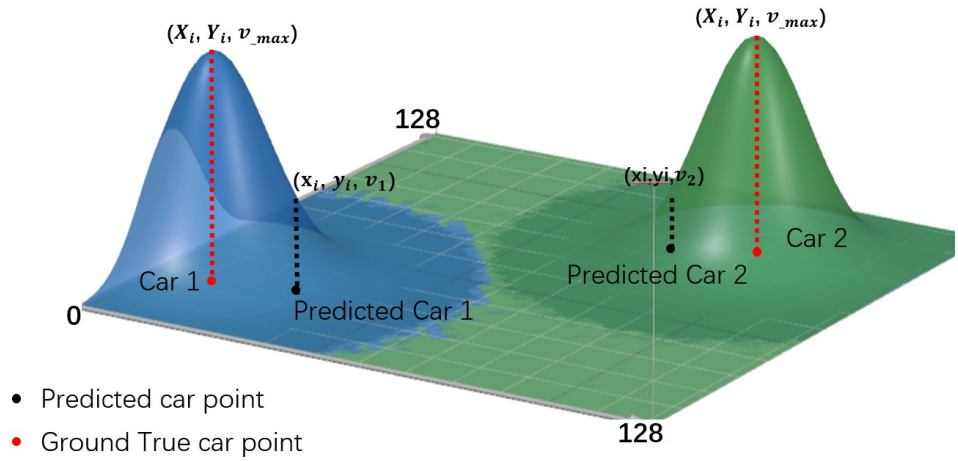


Figure 17: An overview of proposed 2D Gaussian evaluation method. See text for details

4.6 Visualizations

4.6.1 Image visualization

Since the task of this project is visual, tools were developed to evaluate the model's performance visually. An output from this tool can be seen in figure 18. The output contains several figures: the original image, the original image overlayed with detected keypoints, scatter plot of the true keypoints, clustering output of the detected keypoints, Valid detection Hetmap Left/Right, Detected Heatmap L/R and predicted tagmap Left/Right. At the top, there is also information about the loss for this sample, the number of true keypoints, and the number of detected keypoints. The tool contains parameters to select which dataset to visualize to select an image frame and another parameter to select the specific image.

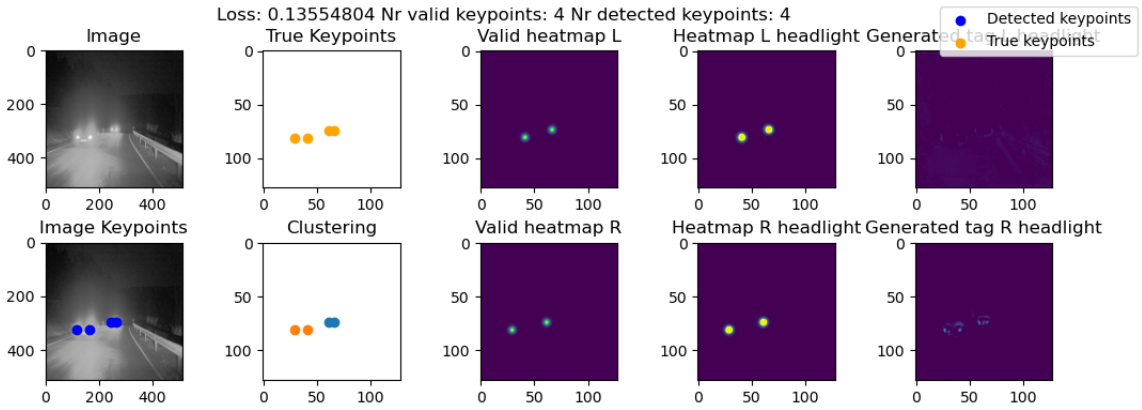


Figure 18: Output from the visualization code

Visualizations are also available during training by using the "–visualizations" parameter, which can be used to evaluate the model while it's training by getting visualizations from the first sample in a batch. The visualizations are updated with each interaction, however this comes at the cost of performance.

4.6.2 Video visualization

Since the PVDN dataset is constructed by taking frames from video sequences it's possible to make multiple predictions of frames that belong to the same sequence and combine them into a video. The predicted keypoints are overlayed on each frame. The keypoints are scaled to match the original resolution of the frame. An example of a frame from the predicted video sequence can be seen in figure 19

Figure 19: Frame from a predicted video sequence

This tool also has parameters to choose which dataset and sequence to visualize. If no sequence is specified then the whole dataset will be visualized.

5 "freedom to operate" analysis

The proposed model has been trained on the PVDN dataset[8], which is licensed under the CC BY-NC-ND 4.0 International License. Consequently, we are free to copy and use it for research purposes.

The inspiration for the model's code was drawn from open-source repositories on GitHub[5]. Therefore, as this project is built upon other open-source code and involves training and validation using publicly available datasets, and in the absence of specific preferences from the project owner, we have decided to adopt the MIT License[9].

The MIT License is concise and permissive, requiring only the preservation of copyright and license notices. Works, modifications, and larger projects can be distributed under different terms, and there is no obligation to provide the source code. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including, but not limited to, the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. Additionally, Our model is designed and implemented in strict accordance with the Project proposal set by the project owner.

6 Conclusions and discussion

6.1 Evaluation results

During the model evaluation, we tested the model's performance using the proposed Gaussian method (refer to the "Solution" section for details on the specific Gaussian method). To showcase the model's performance in multiple situations, we evaluate the results for each case using multiple sigma values. As σ^2 increases, the evaluation method becomes less sensitive, leading to a higher tolerance rate. Conversely, as σ^2 decreases, the scores become more sensitive, resulting in a lower tolerance rate.

Figure20 illustrates an example showing the different output scores corresponding to various σ^2 values when the label is at point 1 and the predicted value is at point 2. The scores range from 0 to 1, where 1 represents the best performance, indicating perfect alignment of the two points. From the figure, it is evident that when the sigma is small, the model's scores are highly sensitive to distance; even a slight deviation of the predicted point results in a significant change in the score.

The covariance matrix of 2D Gaussian distribution

$$\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} \ (\boldsymbol{Scalar} = \boldsymbol{\sigma^2})$$

Example of Gaussian_eval method

Scalar	Euclidean distance	Point1	Point2	eval_value
32	7.07	[60, 60]	[55, 55]	0.457833
	14.14	[60, 60]	[50, 50]	0.043937
	28.28	[60, 60]	[40, 40]	0.000004
64	7.07	[60, 60]	[55, 55]	0.676634
	14.14	[60, 60]	[50, 50]	0.209611
	28.28	[60, 60]	[40, 40]	0.001930
128	7.07	[60, 60]	[55, 55]	0.822578
	14.14	[60, 60]	[50, 50]	0.457833
	28.28	[60, 60]	[40, 40]	0.043937
32*32	7.07	[60, 60]	[55, 55]	0.975882
	14.14	[60, 60]	[50, 50]	0.906961
	28.28	[60, 60]	[40, 40]	0.676634
64*64	7.07	[60, 60]	[55, 55]	0.993915
	14.14	[60, 60]	[50, 50]	0.975882
	28.28	[60, 60]	[40, 40]	0.906961
128*128	7.07	[60, 60]	[55, 55]	0.998475
	14.14	[60, 60]	[50, 50]	0.993915
	28.28	[60, 60]	[40, 40]	0.975882

Figure 20: An example of different σ^2 values and their corresponding scores(eval_value).

Finally, the proposed model underwent Gaussian evaluation, and the resulting score matrix is presented in Table1. Additionally, through the statistical analysis of the model's output on over 4000 images from the test dataset, it was observed that in 68.954% of the images, the model accurately predicted the number of headlights equal to the actual count. Only 0.395% of the images showed the model predicting a higher number of headlights than the actual count, while in the remaining 30.652%, the model predicted fewer headlights than the actual count. The primary reason for this discrepancy may lie in the assumption made during model training regarding the maximum number of headlights per vehicle, where it was assumed that each vehicle has a maximum of two headlights. In reality, some cases involve vehicles with 3 or 4 headlights, leading to the model's underestimation in certain instances.

sigma	score
16	0.68471
32	0.77201
64	0.87345
128	0.93737

Table 1: Resulting score matrix for proposed model

6.2 Visual evaluation

Figure 21: Successful PVDN testset images output by the proposed model.

Figure 22: An instance of headlights being clustered incorrectly

Figure21 displays images output by the proposed model, from these examples, it can be observed that the model's performance meets the expectations. In these and other instances, it can be observed that the detections are generally correct, but there exist instances where the clustering is incorrect. In figure 22 one of these outputs can be seen, this particular example is taken from a video prediction. During the video, there were multiple instances of incorrect clustering.

6.3 Discussion

6.3.1 The impact of n stack

In this project, we employ the stacked hourglass architecture23 introduced by Newell et al[6]. The model's final output is derived as the average of outputs from all hourglass models.

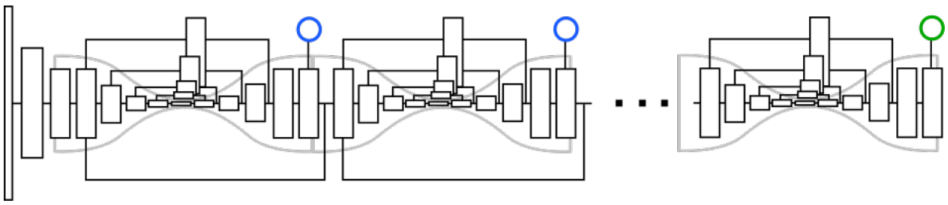


Figure 23: N stacked hourglass architecture[5]

Generally, with a sufficiently large dataset, deeper deep learning models showcase enhanced learning capabilities, resulting in improved accuracy across tasks. In the context of this project, a larger 'n stack' corresponds to superior model performance in headlight detection and clustering. However, it's crucial to note that as the 'n stack' increases, there is a noticeable surge in computational costs, encompassing training time and inference time.

To delve into the relationship between 'n stack' and computational costs, we conducted experiments using a standardized hardware configuration, including an Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz with 32.0 GB of CPU memory, alongside an NVIDIA GeForce RTX 2080 Super with Max-Q Design (Driver Version: 31.0.15.4612, Dedicated GPU Memory: 8.0 GB). The results are illustrated in figure24. From the figure, it becomes apparent that as 'n stack' increases, the training cost of the model experiences exponential growth, particularly in terms of the average training time per batch, accompanied by a noticeable decline in the model's inference speed.

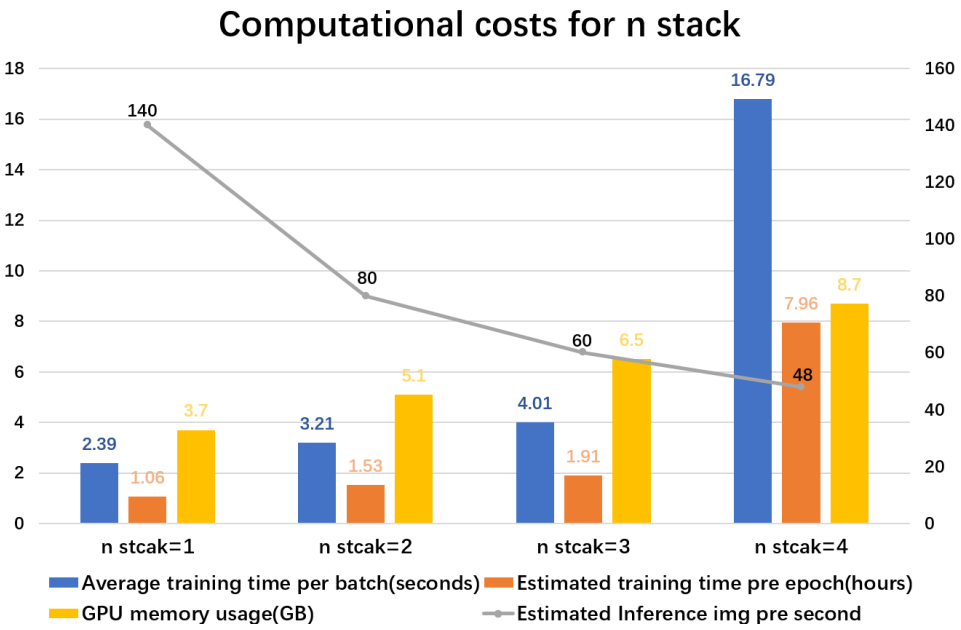


Figure 24: Computation costs for n stack

On the other hand, when 'n stack' is configured as 3, the model secures Gaussian evaluation scores of 0.863 ($\sigma = 64$) and 0.937 ($\sigma = 128$) on the test set. In contrast, when 'n stack' is set to 2, the model achieves comparatively lower Gaussian evaluation scores on the test set, specifically measuring at 0.850 ($\sigma = 64$) and 0.903 ($\sigma = 128$).

Similar to other deep learning projects, in this project, we also face a trade-off between the training cost of the model and its accuracy. Increasing the complexity and depth of the model (increasing 'n stack') may enhance its performance but comes with a significant increase in the use of computational resources. The massive spike in "Average training time per batch" that can be seen in figure 24 is caused by the model size exceeding the GPU VRAM. In the figure, we can see that the model is 8.7 GB while the GPU has a VRAM

of 8GB, when this gets exceeded the GPU needs to page its memory to the RAM which increases computation time.

Therefore, it is crucial to carefully choose the "n stack" parameter during the model design and tuning process to strike a balance between model performance and training efficiency, ensuring optimal results while conserving resources.

6.3.2 Ablation experiments on dynamic thresholding

In this project, we introduced a dynamic thresholding approach for headlight detection, as detailed in Section 4.2.1.

Given the variability and diversity of vehicle headlights, coupled with the complexity of scenes, dynamic thresholding provides significant advantages and flexibility compared to pre-defined fixed thresholding. The influence of fixed thresholding, characterized by varying predefined values, on the model's headlight detection and a comparison of the model results with dynamic thresholding, is depicted in Figure 25. The four tested thresholding values are 0.8, 0.85, 0.9, and 0.95. For the fixed thresholding values higher than these will be considered a detected headlight. In contrast, for the dynamic thresholding, they will be used as the initial thresholding values where most of the detections are removed.

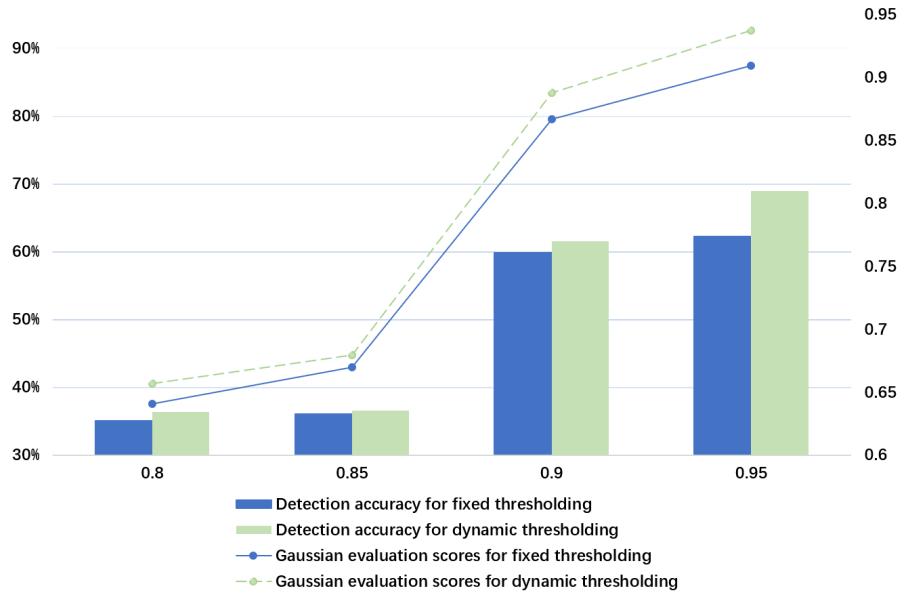


Figure 25: Comparison with dynamic thresholding and pre-defined fixed thresholding

In the figure, predictions are considered successful detections when the model predicts the correct number of headlights in the image compared to the ground truth. Detection accuracy is subsequently calculated as the number of successful detections, divided by the total number of images in the test set.

The graph reveals the substantial influence of the choice of pre-defined fixed thresholding value on the final model's detection accuracy. Detection accuracy, in turn, indirectly affects Gaussian evaluation scores. Dynamic thresholding attains a detection accuracy surpassing 68.9% and Gaussian evaluation scores of 0.937376 ($\sigma = 128$). Compared to fixed thresholding, the proposed dynamic thresholding exhibits an improvement of 10.727% in detection accuracy, with Gaussian evaluation scores improving by 3.306% ($\sigma = 128$).

Hence, both in terms of stability and model performance, the dynamic thresholding method proposed demonstrates superiority over traditional fixed thresholding methods.

6.3.3 Limitations

The proposed model in this project has certain limitations in its application. Despite achieving good accuracy in headlight detection, the model's structure comes with limitations. The number of headlights for a vehicle is predetermined in the architecture of the model and can't be changed after training. In this project, we specified that each vehicle can have a maximum of two headlights (Left and Right), which is not true for all types of vehicles. Trucks for example can have more than two headlights. There also exists a limitation on the maximum amount of vehicle instances during training imposed by the loss function which we chose to be 9. The model can predict more than 9 instances but it will never be shown more than 9 during training. However, having two separate classes for left and right headlights enables the model to detect them independently in situations where only one headlight is visible. Ideally, the model should learn to discern if a headlight is right or left based

on the surroundings. One headlight is visible before the other when going around a curve, depending if it's a right or left curve the left or right headlight will be visible first.

There also exist limitations when it comes to the PVDN dataset, as mentioned previously when adding additional instances where there are multiple vehicles from the day dataset the training data contains 39% no, 39% one, and 22% multiple vehicles. Which means that there still exists an imbalance. Furthermore, Only 6% of the images contain more than two vehicles which means the model sees a few images where there are more than 2 vehicles.

Additionally, the homogeneous nature of the training dataset tailors the model primarily for a specific scenario of vehicle headlight detection. As a result, the model's adaptability to diverse weather conditions, varying climates, different sources of interference, and more complex urban road conditions is relatively limited. Factors like advertisements and other light interferences along city roadsides may adversely affect the model's accuracy.

Furthermore, this project and model aim to test the effectiveness of the method. However, there is a considerable distance to cover before the model can be effectively deployed in practical use. Real-world applications impose higher demands on aspects such as the size of the detection model, detection speed, and accuracy. Addressing these challenges requires further optimization and improvement of the existing model.

In terms of clustering vehicle headlights, the model exhibits moderate accuracy, particularly in situations where two vehicles are near. In such instances, the model tends to mistakenly classify the headlights of two closely positioned vehicles as belonging to the same vehicle. This specific challenge highlights an area that warrants future improvement and enhancement. This might be due to the limitations of representing clusterings in a 1D embedding space, which also limits the push-and-pull mechanics in the loss function. Further refinement is needed to address the intricacies associated with accurately clustering headlights, particularly in scenarios involving closely spaced vehicles.

6.4 Conclusions

In summary, the experimental results validate the effectiveness of the push and pull loss function and the proposed model architecture in the context of headlight detection and clustering. Regarding evaluation metrics, this project introduces a novel Gaussian evaluation matrix tailored for the PVDN dataset to evaluate the model. This Gaussian evaluation method allows for the adjustment of sensitivity in the evaluation process by inputting different σ values, catering to diverse requirements.

The Gaussian evaluation, along with the resultant score matrix and visual output examples, demonstrates the overall performance of the model, aligning with anticipated outcomes. Additionally, the model, evaluation matrix, and related visualizations proposed in this project fulfill the requirements outlined in the initial Project Proposal.

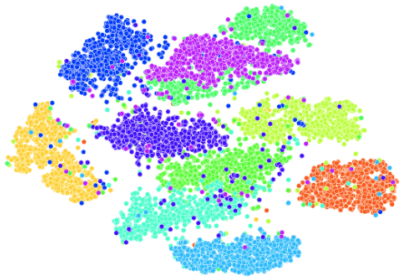
Although our results look promising it's difficult to truly know the effectiveness of the methods proposed using the PVDN dataset. Since the already mentioned instance imbalance in the dataset exists in the validation and test sets. Further work should look into finding an alternative dataset.

In conclusion, the contributions of the proposed model, evaluation matrix, and associated visualizations are in harmony with the project owner's expectations, as outlined in the initial project proposal. The developed deep learning framework, featuring the integration of push and pull losses, establishes itself as a viable solution for achieving accuracy and sensitivity in headlight detection and clustering.

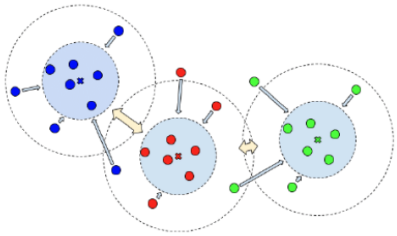
7 Work expansion

To address some of the limitations in the approach and explore the feasibility of the push and pull method for headlight detection and clustering, we expanded our work. Based on the reference paper "Semantic Instance Segmentation with a Discriminate Loss Function" [10], we modified our existing architecture and introduced a new loss function. This required changes to the data loader, output parsing, and evaluation methods. However, the goal was to use as much code from the first approach as possible to complete the implementation in time. Figure 26 depicts the three main areas addressed by the new approach: No limits on headlights per vehicle, a higher dimensional embedding space to represent the clustering, and using a more traditional approach to push-and-pull loss.

(a) No limit on headlights per vehicle.



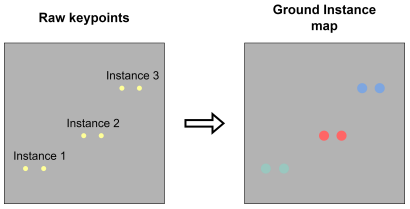
(b) Higher dimensional embedding space.



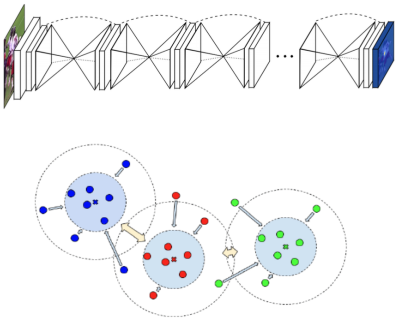
(c) A more traditional approach to push-and-pull loss.

Figure 26: Things to address in the new approach.

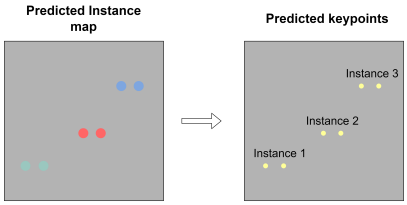
The task in the reference paper is instance segmentation using a high-dimensional embedding space. From this, we formulated a basic pipeline for this approach. First, we convert the PVDN coordinates to instance segmentation maps that can then be used for training a model. When parsing the output of the model we still want coordinates, to achieve this we use the predicted instance segmentation map to extract headlight coordinates. A visual representation of this pipeline can be seen in figure 27.



(a) 1. Converting coordinates to instance map.



(b) 2. Using instance maps to train the model.



(c) 3. Convert predicted instance map to headlight coordinates.

Figure 27: The three main steps of the new approach

7.1 Architecture

To match the approach in [10] we changed our architecture to have three outputs: Semantic segmentation map (1 dimensional), instance embedding space (32 dimensional), and instance count prediction (1 Value). Figure 28 shows these three outputs being sampled from a 64-dimensional output from the stacked hourglass model, this happens for each stack. The model uses 3-stacks as in the previous approach. The three outputs are used to parse the final instance segmentation map. During testing, it was found that the instance count prediction had

low accuracy which caused issues when parsing the instance segmentation map. The instance count prediction is used for clustering the embedding space with KMeans clustering. KMeans clustering needs to have information about how many clusters to find. DBScan was implemented as an alternative clustering method that does not require the number of clusters to detect. Thus the instance count output could be removed, simplifying the model and loss function.

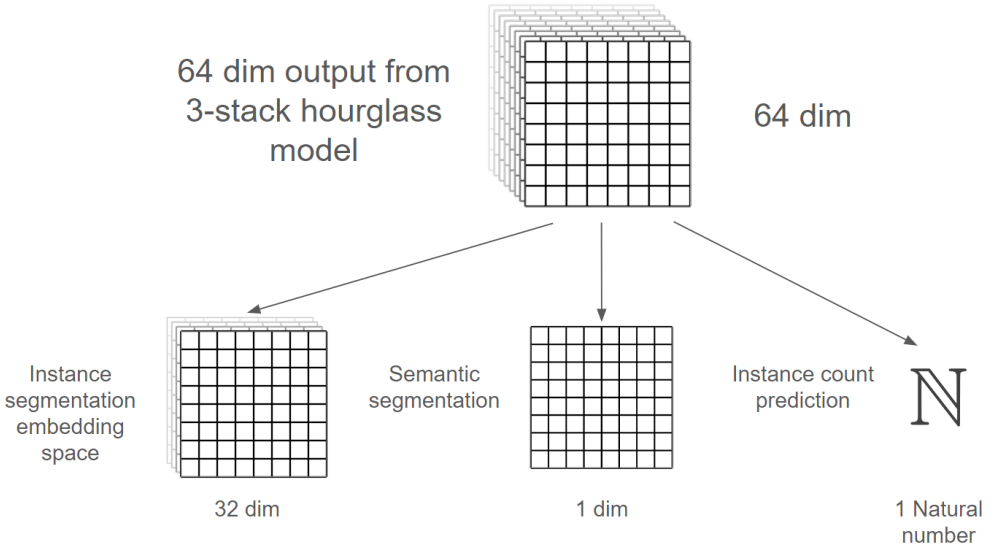


Figure 28: Output from the 3 stacked being parsed to the three outputs

7.2 Loss function

For the two outputs, semantic segmentation and embedding space, separate loss functions are applied: binary cross-entropy loss (BCE) and push-and-pull loss. These loss functions are applied to each stack, in our case that means three of each loss are then combined into one loss value.

7.2.1 Semantic segmentation - Binary Cross Entropy (BCE)

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

Since the semantic segmentation map is a 1D output, binary cross entropy is used to quantify the difference between the true map and the prediction.

7.2.2 Instance Embedding Space - Push-and-pull

The push-and-pull loss consists of three different terms: variance, distance, and regularization. To calculate these terms the mean of each instance is calculated represented by squares in figure 29.

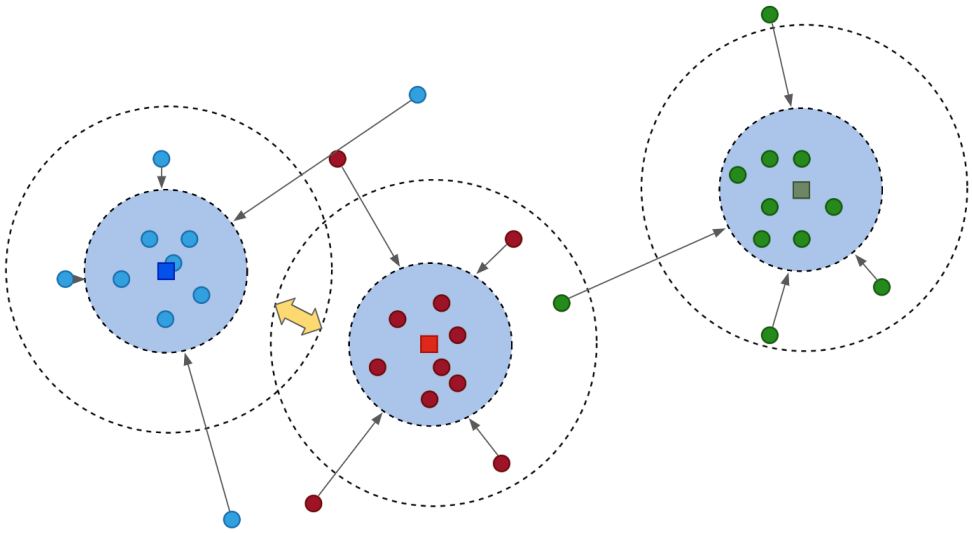


Figure 29: Example of push-and-pull loss

xThe variance term pulls points of the same instance toward the mean, this pull force however is limited to a certain radius around the mean to reduce overfitting and unnecessary movement in the embedding space [10]. This reduces the variance within the instance. The distance term pushes away instance means that have an intersecting push radius. In figure 29 it can be observed that the blue and red points have intersecting radii which means that they have to be pushed away from each other. The push force is not applied to the green mean since it doesn't have any intersections with another instance mean. To keep the instance points centered in the embedding space a regularization term is applied pulling points toward the center. Especially in the training of models and the computation of loss functions in high-dimensional spaces, regularization proves particularly effective[11].

7.3 Data loader

The dataset used in this extended work is the same as that of the earlier approach which is explained in 2.3. Raw image data from the PVDN dataset are converted into a format suitable for our model. This involves converting key point information into segmentation maps. This transformation is crucial for both amplifying the accuracy of headlight detection and enabling efficient clustering. The process begins by interpreting each keypoint, representing a headlight's location, from the PVDN dataset.

For each headlight keypoint, an adaptive representation is created. This includes determining the size and shape of the representation based on factors like the distance between headlights in the same vehicle and the intensity of the light from the headlight. The intensity or brightness score is higher if the number of bright pixels around the key point is greater. The size of each headlight representation on the instance map is regulated according to its deemed brightness and distance. Brighter and closer headlights are represented with larger sizes compared to dimmer or distant headlights. When there is only one headlight in a vehicle, only the brightness score is considered. After considering distance and brightness scores, the radius is assigned to each headlight that adheres to the minimum and maximum radius thresholds. These circular areas distinctly mark the position and size of each headlight, effectively differentiating them from the background. Figure 30 shows an image and respective target segmentation map which is an input to the model training.

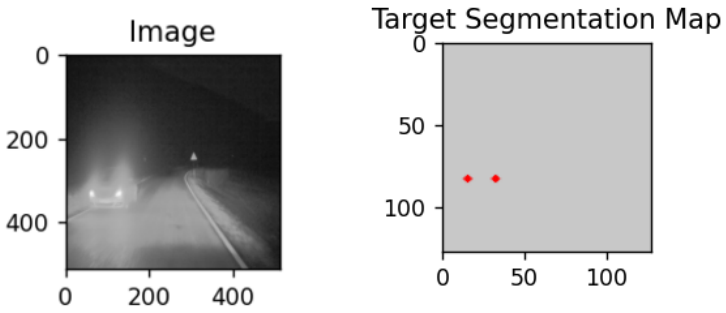


Figure 30: Image and its Segmentation map

7.4 Output parsing

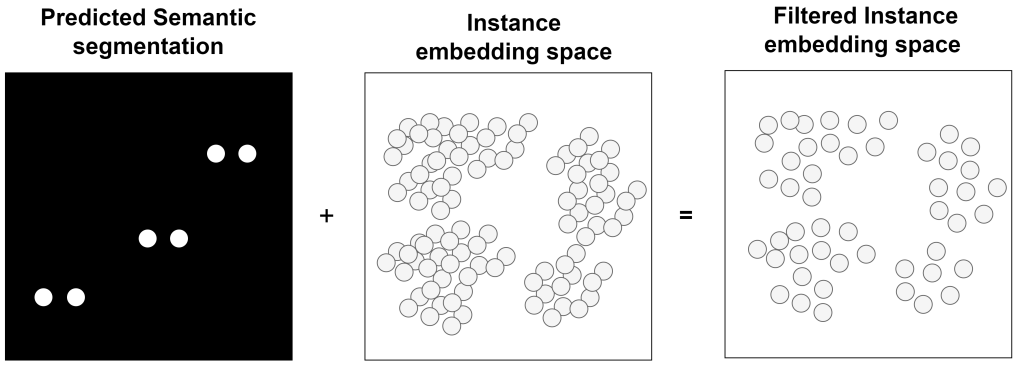


Figure 31: Filtering the embedding space using the semantic segmentation map

The predicted semantic segmentation map and embedding space are used to parse the final instance segmentation map. The semantic segmentation map segments out the headlights without taking the instance into account. Figure 31 shows how the semantic segmentation map is used to filter the embedding space which only leaves points that contain headlights.

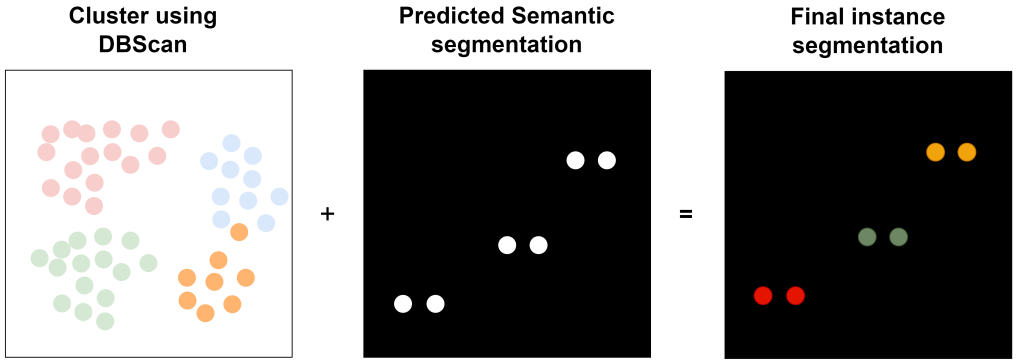


Figure 32: Using the semantic map and clustered embedding space to get the final instance segmentation

The remaining points are clustered using DBScan. The final instance segmentation map is constructed by selecting points in the clustered embedding space using the semantic segmentation map which can be seen in figure 32. This is done by selecting each point containing a headlight in the semantic segmentation map to find the corresponding point in the clustered embedding space and retrieving its cluster value.

7.5 Key point parsing

In the extraction of key points from the output of the model, the approach differs depending on the characteristics of the detected blobs, which represent potential headlights of vehicles. When the blob's width and height are almost equal, it is assumed to be a single headlight and the centroid of the blob is considered the keypoint representing the headlight. If the width significantly exceeds the height, It can be considered as a combination of two headlights. K-means is employed to segment the big blog into two small blobs and the centroid of each blob is considered as keypoints.

7.6 Evaluation

To evaluate the new method we used Intersection over Union (IoU). Intersection over Union is a more traditional way to evaluate segmentation performance and it can use the semantic segmentation maps that are generated with the new approach.
The average IoU score with the new model is around 0.5. Generally, this score is considered a "good" score for object detection. In our case, the score seems to be lower compared to the actual results that we observe from video sequence visualizations and segmentation maps. The issue here might be the segmentation areas that the model produces, as well as the "ground truth" segmentation for detecting the vehicles are quite small, compared to the whole size of the image, therefore the actual "union" and "intersection" that is calculated can

have less overlap, keeping in mind, that we generate the true segmentation maps from the keypoint coordinates in the dataset annotations.

7.6.1 Visual evaluation

The visualization was remade to represent the input and output of this new model. The most notable addition is the embedding space visualization which can be seen in 33. To visualize the embedding space t-SNE is used to reduce the 32 dimensions into 2 dimensions.

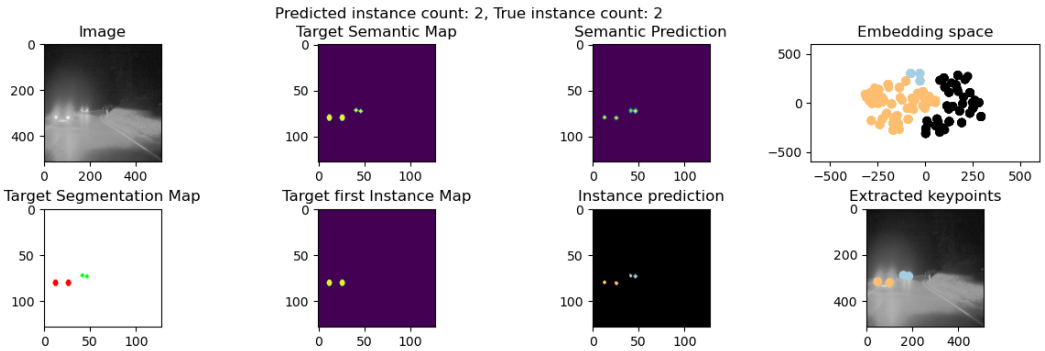
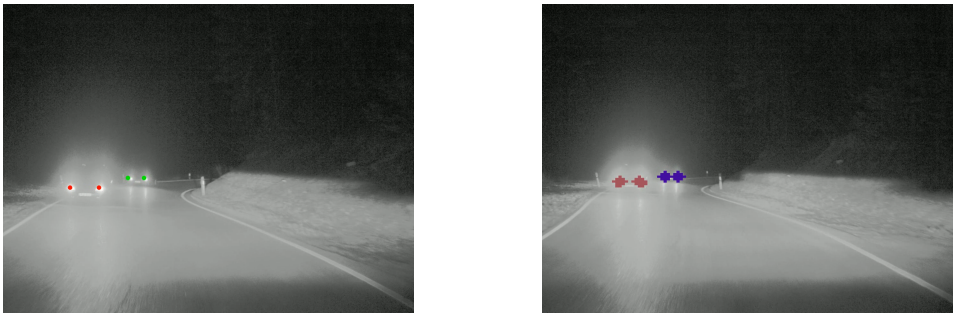


Figure 33: Output from the new visualization tool.

Similarly to the previous approach video visualizations were implemented for the new model as well. As mentioned the previous approach had issues with clusters jumping between vehicles when inspecting video visualizations using the new model these rarely occur. Because the keypoint parsing adds another possible point of failure it's possible to visualize the raw instance embeddings in the video both options can be seen in figure 34.



(a) Frame from video overlayed with parsed keypoints

(b) Frame from video overlayed with raw instance predictions

Figure 34: Video frames from the new approach

7.7 Conclusion

By addressing the technical limitations of our previous approach we were able to improve the clustering performance. The biggest performance gain probably came from using a higher dimensional embedding space allows the model to have a higher capacity for representations that can capture more complex representations. This is in contrast to the previous approach where each pixel in the predicted tag map could be seen as a 1D embedding space. The more traditional embedding space also meant that we could use ready-made clustering methods such as KMeans and DBscan, reducing the complexity of having to implement a custom clustering solution.

When parsing the output there is no need to specify a threshold for detections, instead, the semantic segmentation map is a learnable regression head that requires no user-specified thresholds. There is however one added complexity in the conversion from coordinates to instance segmentation maps used for training and then during output parsing when converting the predicted instance map back into coordinates. Even though the coordinate parser works well in our tests it still introduces another point of failure.

This approach and the previous one share the same issue of dataset imbalance due to the PVDN dataset. The same mitigation techniques of using weights and samples from the day dataset are used for this approach,

but as mentioned previously this still isn't enough to address the fact that only 6% of the images contain more than two vehicles. Further work should look into another dataset to get a fair assessment of our two approaches.

References

[1] Tuan-Anh Pham and Myungsik Yoo. Nighttime Vehicle Detection and Tracking with Occlusion Handling by Pairing Headlights and Taillights. *Applied Sciences*, 10(11):3986, January 2020. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[2] Sungmin Eum and Ho Gi Jung. Enhancing Light Blob Detection for Intelligent Headlight Control Using Lane Detection. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):1003–1011, June 2013. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[3] Michael Angelo D. Ligayo, Michael T. Costa, Ryan R. Tejada, Luisito L. Lacatan, and Christopher Franco Cunanan. An Augmented Deep Learning Inference Approach of Vehicle Headlight Recognition for On-Road Vehicle Detection and Counting. In *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pages 389–393, March 2021.

[4] Lukas Ewecker, Ebubekir Asan, Lars Ohnemus, and Sascha Saralajew. Provident vehicle detection at night for advanced driver assistance systems. *Autonomous Robots*, 47(3):313–335, March 2023.

[5] Alejandro Newell, Zhiao Huang, and Jia Deng. Associative Embedding: End-to-End Learning for Joint Detection and Grouping, June 2017. arXiv:1611.05424 [cs].

[6] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pages 483–499. Springer, 2016.

[7] Lars Ohnemus, Lukas Ewecker, Ebubekir Asan, Stefan Roos, Simon Isele, Jakob Ketterer, Leopold Müller, and Sascha Saralajew. Provident vehicle detection at night: The pvdn dataset, 2021.

[8] Provident Vehicle Detection at Night (PVDN).

[9] Jerome H Saltzer. The origin of the "mit license". *IEEE Annals of the History of Computing*, 42(4):94–98, 2020.

[10] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function. *arXiv preprint arXiv:1708.02551*, 2017.

[11] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269, 1995.