CSC420 Final Project Report

Tianxi Hu (hutianx1, 1004715178), Esmat Sahak (sahakesm, 1004928126)
Project Repository: https://github.com/esmatsahak/CSC420-Project

1 Introduction

According to a report released by McKinsey & Company, the autonomous driving industry is projected to generate \$300-\$400 billion in revenue by 2035 [1]. This is nearly 10 times the current projected revenue of \$40-55 billion [1]. The recent focus on autonomous vehicle research is evident from Figure 1, which illustrates the sharp increase in autonomous driving publications from 2000-2017.

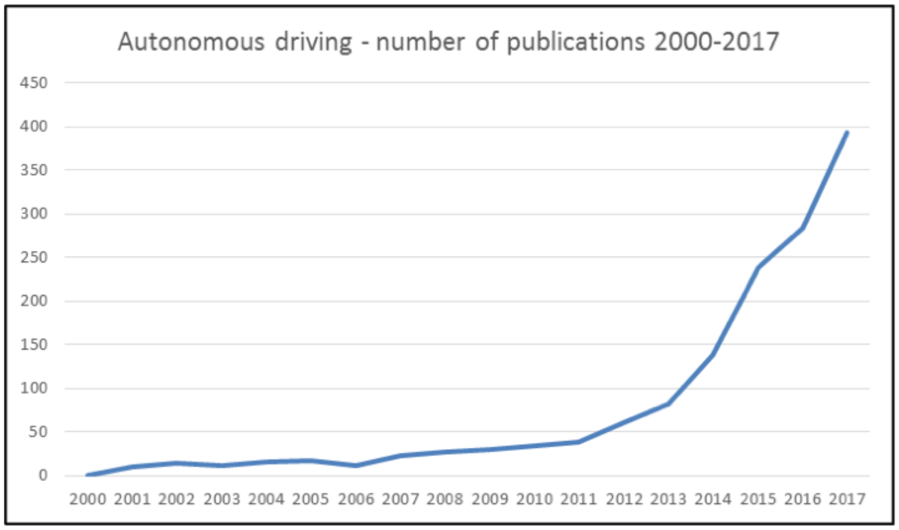


Figure 1: Number of autonomous driving research papers on the Web of Science from 2000-2017 [2].

This project covers concepts in autonomous driving, particularly extracting information given a set of parallel stereo highway images taken by a left and right camera on a moving vehicle. This data was collected from the KITTI Vision Benchmark Suite [3], one of the most popular datasets used for computer vision tasks. An overview of the data is provided in Table 1.

The primary tasks of this project include road detection and vehicle detection. There are related subtasks for each task, such as estimating the ground plane and drawing 3D bounding boxes to represent detected cars in world-coordinates. A more detailed version is given below:

- **Preliminary tasks:** compute and visualize disparity and depth maps of an input image (completed by Hu)

- **Road classifier:** train a supervised classifier to detect road pixels, visualize predictions on test set by marking road pixels a distinctive colour (completed by Hu)
- **Ground plane:** approximate the ground (road) plane of an image given the predicted road pixels, visualize results in 3D point cloud (completed by Sahak)
- **Car detection:** use a pre-trained model to detect cars in an image, visualize results by drawing 2D bounding boxes around detected cars (completed by Sahak)
- **Viewpoint classifier:** train a classifier to determine car orientation, visualize predictions by drawing a viewpoint vector (completed by Sahak)
- **3D bounding box:** compute 3D bounding box coordinates of detected cars, visualize results in 3D plot with estimated ground plane (completed by Sahak)

Directories	Description
data/train/calib data/test/calib	Contains text files of camera calibration parameters (e.g. focal length, location of camera, etc.)
data/train/image_left data/test/image_left	Contains image files taken by left camera
data/train/image_right data/test/image_right	Contains image files taken by right camera
data/train/gt_image_left	Contains ground truth segmentation mask of train images taken by left camera
data/train_angle/image	Contains image files of highway for viewpoint classification task
data/train_angle/labels	Contains binary MATLAB files of 2D bounding box and orientation of cars in viewpoint train images

Table 1: Overview of project data extracted from KITTI dataset

The next sections cover these tasks in more detail, describing the methods applied to solve these tasks and their corresponding results.

2 Preliminary Tasks

The preliminary tasks involve writing Python functions to compute and visualize the disparity and depth maps. The Semi-Global Block Matching (SGBM) algorithm [4] is used to determine the disparity map. From the disparity map, the depth map is simple to compute provided the focal length (*f*) and baseline (*T*) from the calibration files.

2.1 Disparity Map

The Semi-Global Block Matching (SGBM) algorithm is used for computing the disparity map between a pair of stereo images. The algorithm consists of the following steps:

- **Preprocessing**: The left and right images are first preprocessed to reduce noise and improve matching. This may involve techniques such as smoothing, histogram equalization, or edge detection.
- **Matching Cost Computation**: For each pixel in the left image, a matching cost is computed by comparing the corresponding block of pixels in the right image. The matching cost measures how similar the two blocks are, and is typically based on the sum of absolute differences or the sum of squared differences.
- **Disparity Search**: Once the matching costs have been computed, the algorithm searches for the disparity value that minimizes the cost. This search is performed over a range of possible disparity values and the best disparity value is selected based on the minimum matching cost.
- **Disparity Refinement**: To improve the accuracy and consistency of the disparity map, the algorithm performs a semi-global optimization step. This involves computing a cost volume that contains the matching costs for all possible disparity values at each pixel. The cost volume is then smoothed using a penalty function that encourages neighboring pixels to have similar disparities. The final disparity map is obtained by selecting the disparity value that minimizes the total cost along a path through the cost volume, which is determined by the penalty function.

We use the ***StereoSGBM_create()*** function in OpenCV [5] for a convenient implementation of the SGBM algorithm. It takes the left and right images as input, as well as several parameters that determine the properties of the stereo algorithm, which includes *num_disparities* parameter specifies the maximum number of disparities to search for, *block_size* parameter determines the size of the matching window and *min_disparity* parameter specifies the minimum disparity value. The disparity map is normalized and saved as a grayscale image.

Appendix A contains the implementation of the ***computeDisparity*** function accepting left and right images and the parameters of the ***StereoSGBM_create()*** function as input, and returns the disparity map as output. Figure 3 illustrates the disparity map for the image *umm_000000.jpg* (Figure 2) given the parameters *num_disparities* = 64, *block_size* = 9, and *min_disparity* = 1. The parameters were selected after some trial and error and inspecting whether the disparity and depth maps aligned with expectation (explained later). A darker shade represents a lower disparity value.

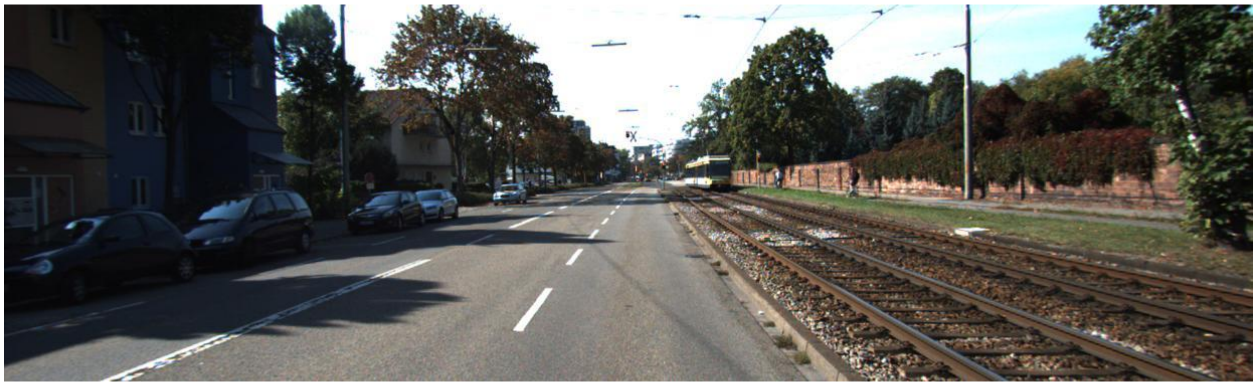


Figure 2: Image umm_000000

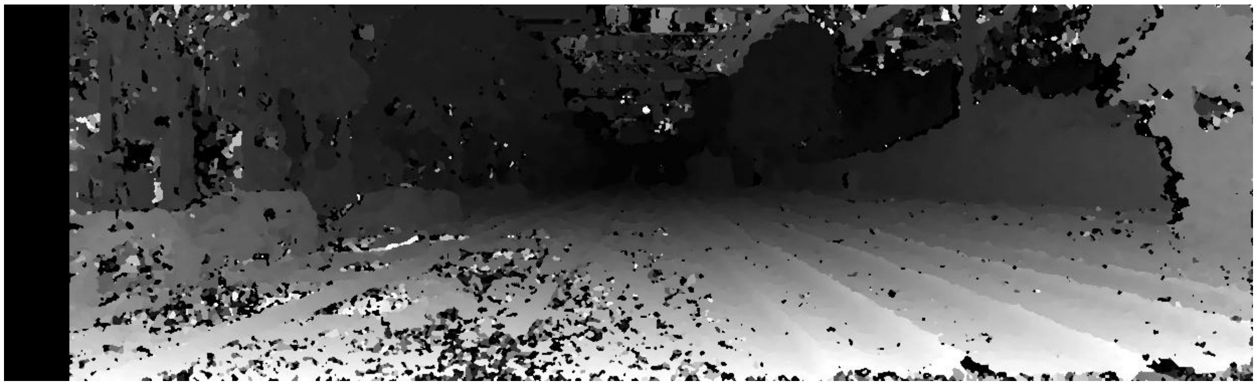


Figure 3: Disparity map of image umm_000000

The expectation for the disparity map is that as you move closer towards the horizon or vanishing point, disparity values should decrease. For the most part, this is observed to be the case as the shade of the pixels gets increasingly darker where the road appears to converge in the image. The dark pixels on the left edge of Figure 3 is a result of those pixels not being captured in the right image.

2.2 Depth Map

The depth of each pixel is computed by taking the disparity map as input, as well as the camera baseline distance *T* and the camera focal length *f*. Camera parameters are read from a calibration file (see ***getCameraParams*** function in Appendix A). This file contains the focal length and the baseline distance of the stereo camera, which are used to compute the depth map using the formula below. The resulting depth map is normalized and saved as a grayscale image.

$$depth = \frac{f * T}{disparity}$$

The function ***computeDepth*** in Appendix A accepts the disparity map, *f*, and *T* and returns the output depth map. Figure 4 illustrates the depth map for image *umm_000000.jpg*, a darker shade implies a lower depth. Similar to the disparity map, the depth should increase as you converge to the vanishing point. This is evident in Figure 4, as the regions near the convergence point are of a lighter shade.

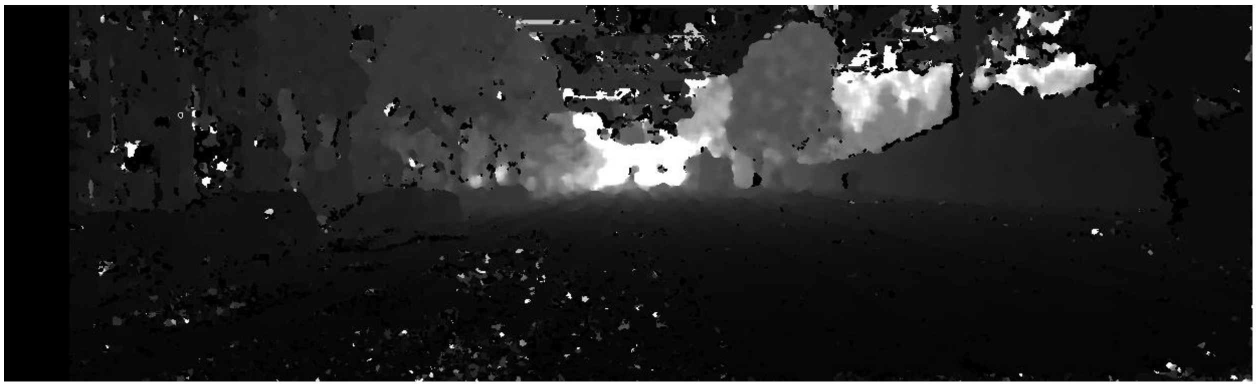


Figure 4: Depth map of image umm_000000

3 Road Classification

Before a classifier is trained to learn road detection, a featurization method must be devised to represent pixels as vectors. Once this is done, standard classification methods can be applied.

3.1 Features

The goal of devising features is to simplify the representation of the image. Creating a feature vector for each pixel is inefficient, given each image consists of 465,750 pixels (1242 × 375). Instead, we group similar pixels together by computing superpixels. Superpixel segmentation clusters similar pixels together to form larger segments that have similar properties, which reduces the number of pixels that need to be processed. The SLIC algorithm [6] is applied here for superpixel segmentation, as it is fast and produces high-quality segmentations.

The ***getFeatures*** method in Appendix B shows that each image was segmented into 1000 superpixels. Figure 5 visualizes these superpixels for image *umm_000000.jpg*. To further reduce computation, the centroid of each superpixel was assumed to be representative enough to gauge whether the superpixel is part of the road or not. Although taking the average feature value over all the pixels inside a superpixel would probably result in better results, it is far more computationally expensive.

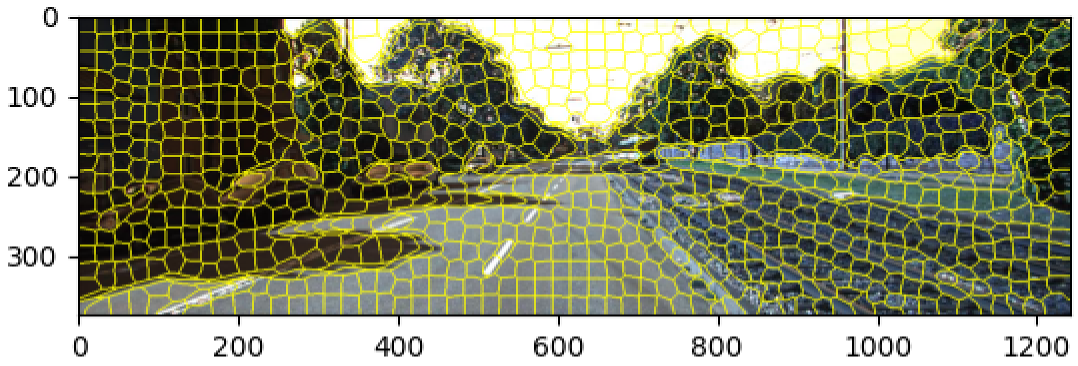


Figure 5: Superpixels of image umm_000000 after SLIC segmentation

Features that are most indicative of whether a pixel is part of the road or not include position and color. The 2D and 3D positions of the centroid, along with the RGB values of the centroid were selected to form a 8×1 feature vector for each superpixel. The 3D position (X, Y, Z) can be computed from the depth (Z), 2D position (x, y), and camera parameters (f, p_x, p_y) where (p_x, p_y) is the principal point where the optical axis intersects with the image plane. These formulas are listed below. Note that when plotting on a 3D plane, the depth is the y-axis, so the point is actually *(X, Z, Y)*.

$$X = \frac{Z(x - p_x)}{f} \qquad Y = \frac{Z(y - p_y)}{f}$$

The justifications for these feature choices are provided below.

- **2D centroid position** (x, y): The center of the superpixel is a useful feature to identify road segments, as they typically have a horizontal orientation and are located in the bottom part of the image.

- **Color of centroid** (*RGB*): Road segments typically have a specific color range, and hence, the color of each superpixel can be used to differentiate between road and non-road segments.
- **3D centroid position** (*X, Y, Z*): The road surface forms the ground plane, so points that lie on the ground plane are likely to be on the road. For this, the 3D position is required.

The code for generating these features is shown in the ***getFeatures*** function in Appendix B. It reads the left and right images, extracts relevant camera parameters from the calibration file, computes the depth map, and uses the SLIC algorithm to segment the left image into 1000 superpixels. For each segment centroid, the features listed are computed and a list of feature vectors is returned.

Normalizing these features was considered (to train a classifier independent of image size), but it is difficult to normalize the 3D position as there is no known bound on these values. Passing in unnormalized values shouldn't present much of a problem as all images are the same size, and all values are of similar scale. In future iterations, normalization methods will be investigated to improve the classifier.

3.2 Multilayer Perceptron Classifier

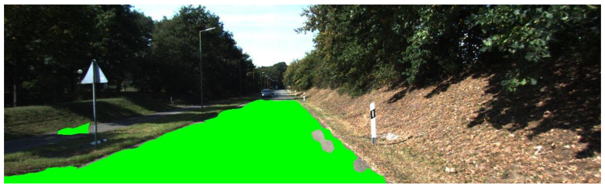
A multilayer perceptron (MLP) classifier is then trained given the feature vectors and their corresponding ground truth values extracted from the provided segmentation masks. It is a good choice for this task as it has a hidden layer that enables it to learn complex non-linear relationships between the input features and the output labels. It also has the ability to handle large datasets for image classification, which makes it suitable for this task. Additionally, MLPs are a commonly used and well-studied neural network architecture, which makes it easier to find relevant documentation and existing implementations. In our case, we used the implementation of ***MLPClassifier*** from ***sklearn.neural_network*** [7] because classifiers from the *sklearn* package have low latency (i.e. train faster) compared to other classifiers from other packages [8]. Relevant classification code can be found in the ***classifier*** method in Appendix C.

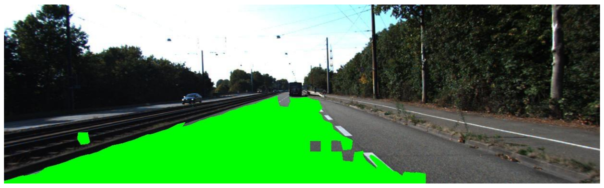
The training data is split into a train and validation set (80%-20% split respectively), in order to assess performance on a held-out set (test data doesn't have ground truths). Table 2 summarizes the performance metrics on the validation set. The model achieves an overall accuracy of 95%, and 86% when just taking into account road pixels. Precision and recall scores for the road pixels are 0.93 and 0.80, which are also acceptable. It is important to verify these scores, as an accuracy of 80% can be achieved on the entire dataset if all pixels are said to not be road pixels.

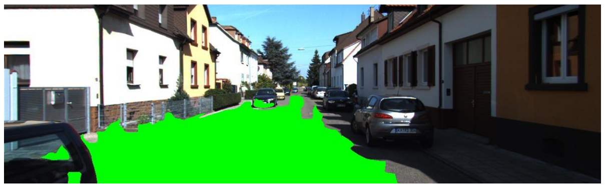
Dataset	Precision	Recall	Accuracy	Examples
Not Road Pixels	0.95	0.99	0.97	20,243
Road Pixels	0.93	0.80	0.86	4,841
All Pixels	**0.95**	**0.95**	**0.95**	**25084**

Table 2: Classification performance metrics of road classifier

Figure 6 visualizes how well the model generalizes to the test set, by marking the predicted road pixels in green. The ***markRoadPixels*** method in Appendix C accomplishes this task. Overall, it performs well, but struggles classifying road pixels on the right-hand side of the image (see Figure 6b and 6d). It also struggles classifying road pixels in between cars because they are darker due to shadow effects (Figure 6c). This shows there is room for improvement in feature selection, particularly incorporating features that de-emphasize position and color, such as texture or gradients.


(a) uu_000012


(b) umm_000069


(c) uu_000027

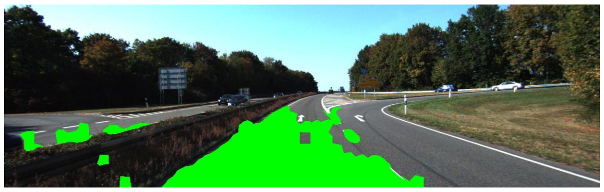

(d) umm_000093

Figure 6: Road detection predictions of sample test set images with MLP classifier

3.3 U-Net Classifier

Generally, deep learning classifiers tend to outperform machine learning classifiers, especially in domains with large amounts of data [9]. To test this hypothesis in the context of road detection, an Attention U-Net [10] classifier was developed. The model was chosen for its use of *attention gates* (AGs) which implicitly learns to suppress irrelevant regions in an input image while highlighting salient features useful for a specific task, in our case the segmentation of roads.

The training data is split into a train and validation set (80%-20% split respectively). The Dice score between the prediction road pixels and the label road pixels is used as the metric to measure the model's performance on the validation set. Given two sets, X and Y, the Dice score (DSC) is defined as:

$$DSC = \frac{2|X \cap Y|}{|X|+|Y|}$$

The model is trained over 50 epochs with dice loss and a dropout of 0.1 (code in Appendix D). Figure 7 shows a plot of the training and validation loss, as well as the DSC at each validation epoch over the last 10 epochs. Figure 8 visualizes the prediction with a comparison of the original image, given label and model prediction for several images from the validation set. The best mean Dice score achieved on the validation set is 0.8747. Overall, the model is able to produce predictions that align with the labels very well (Figure 8a). However, it is prone to predicting false negatives under the influence of lane markings (Figure 8b) and false positives with large areas of shadows (Figure 8c).

The deep learning model is able to generate better predictions compared to the MLP classifier and does not require feature selection. However, it is computationally expensive and takes approximately 1 hour / 10 epochs to train, so the outputs take much longer to generate. As a result, the outputs from the MLP classifier are used in the following sections.

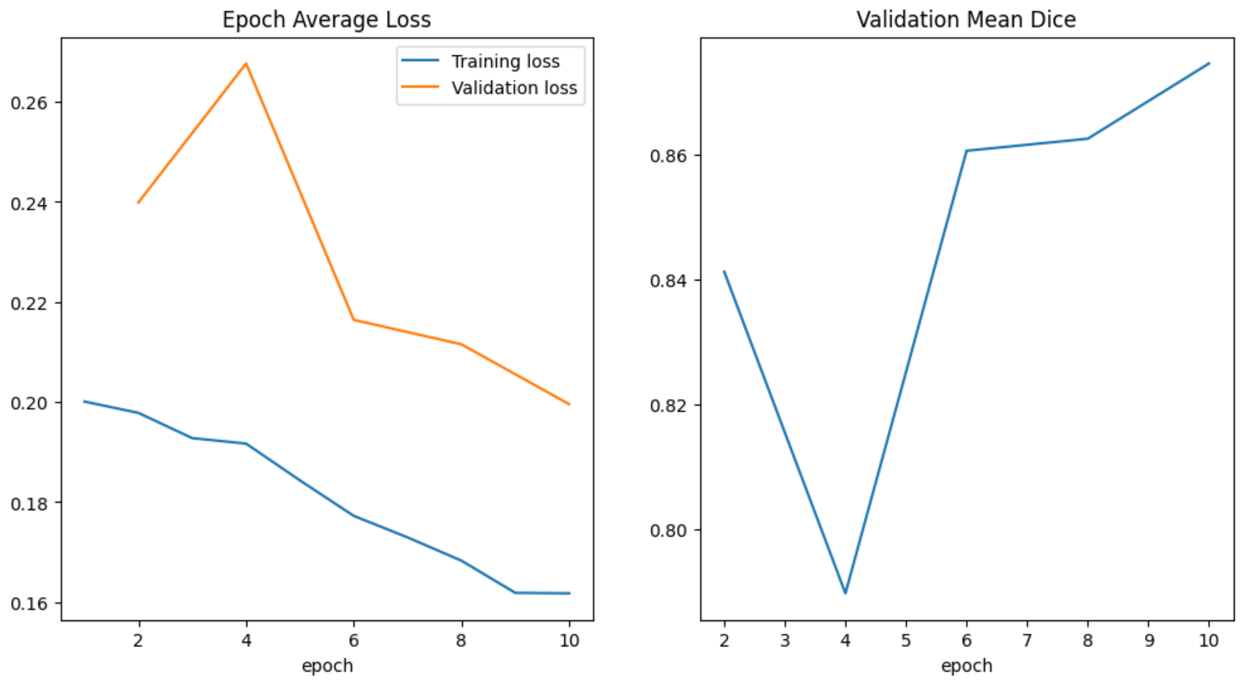
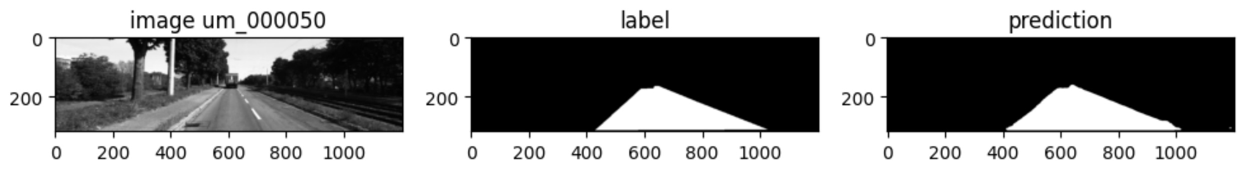
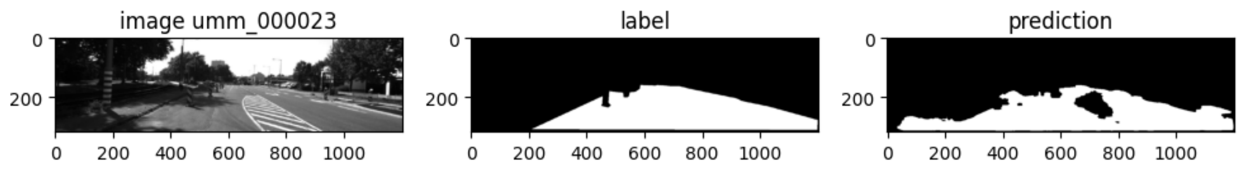


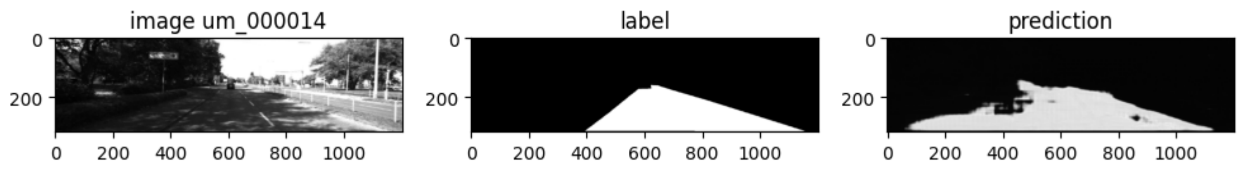
Figure 7: Training and validation loss with mean validation dice score over epoch 40 to 50



(a) um_000050



(b) umm_000023



(c) um_000014

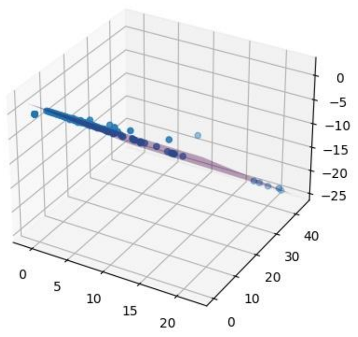
Figure 8: Road detection predictions of sample test set images with Attention U-Net

4 Ground Plane

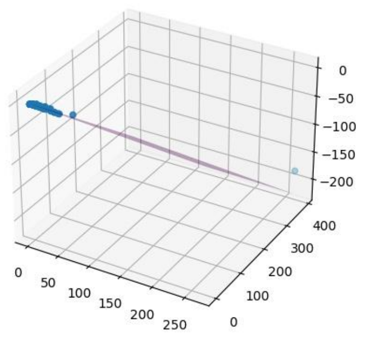
To fit a plane to the predicted road pixels (i.e. test set predictions from MLP classifier), the centroids of predicted road superpixels of each test image are converted to 3D points (see ***preprocessRoadPixels*** in Appendix E). Again, superpixel centroids are used to reduce computational cost. The same method presented in Section 3.1 is used to get the 3D points.

To ensure the estimated plane is robust to outliers, the random sample consensus (RANSAC) algorithm introduced in lecture was applied. To estimate a plane, RANSAC takes 3 non-collinear points, computes the equation of the plane, then determines the number of points within some threshold distance of the plane (i.e. inlier count). RANSAC chooses the plane that maximizes the inlier count, hence reducing the effect of outliers. The ***sklearn.linear_model*** package offers ***RANSACRegressor*** [11], a regression classifier that accomplishes this estimation process (see ***getBestPlane*** function in Appendix E).

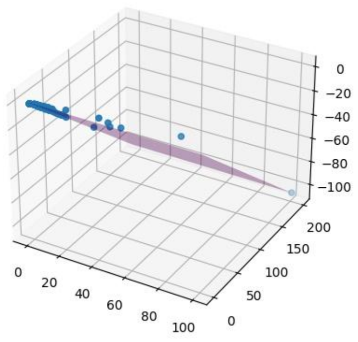
Figure 9 plots the centroid of road superpixels (blue) and the estimated road plane (purple) of some sample test images. The ***plot3DPoints*** function in Appendix E generates these visualizations. The plots demonstrate that the estimation algorithm performs exceptionally, and with an average inlier acceptance rate of 87.4%. Although the results are promising, some of the planes are too steep to be a ground plane, highlighting some inaccuracies which likely stem from the depth estimations. Figure 10 illustrates the 3D point cloud for a sample test image and the estimated plane (red). Python's Open3D library [12] was used to generate this visualization, and the code is accessible in Appendix F in the ***convertToPointCloud*** method. The side view illustrates that the plane diverges from the road, as you move further away from the vanishing point. This aligns with the plots in Figure 9, where there is a concentration of points at the origin, which is the vanishing point. This means that the MLP road classifier struggles at detecting pixels further away from the vanishing point. The front view shows the extent of divergence, which is a considerable margin.



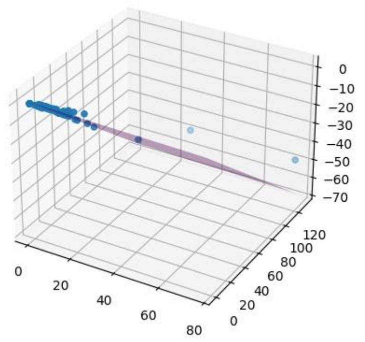
(a) uu_000012



(b) umm_000069

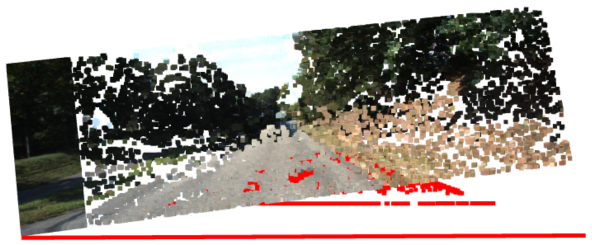


(c) uu_000027

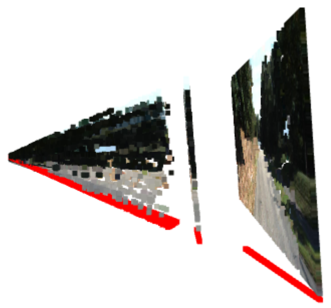


(d) umm_000093

Figure 9: Estimated ground plane of sample test images



(a) Front view



(b) Side view

Figure 10: 3D point cloud of test image uu_000012 and estimated ground plane (red)

5 Car Detection

This task is relatively straightforward as it leverages a pre-trained model (i.e. only inference needed), Faster R-CNN [13]. On a high-level, R-CNN models first identify regions of interest (i.e. objects), then pass these regions to a CNN-based classifier to label the object class. Faster R-CNN eliminates the need for step 3 in Figure 11 by using the same feature map for the image for both identifying and classifying regions (Figure 12).

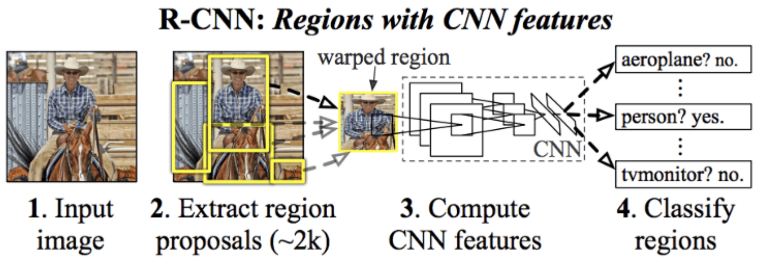


Figure 11: High-level overview of R-CNN architecture [14]

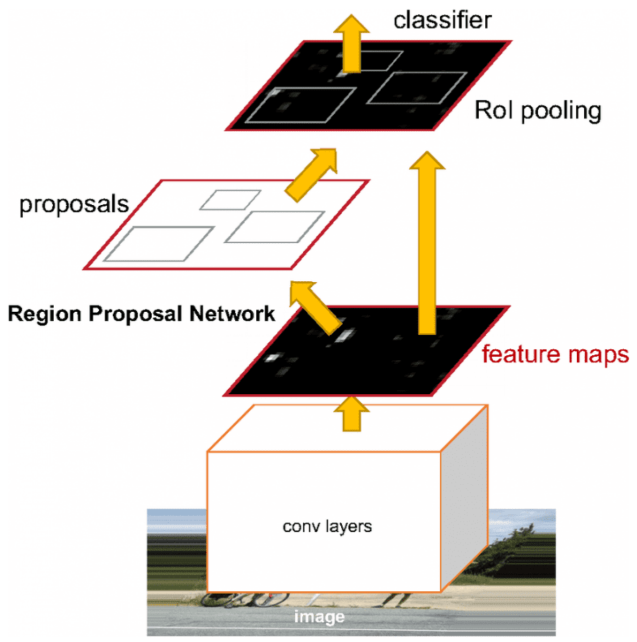
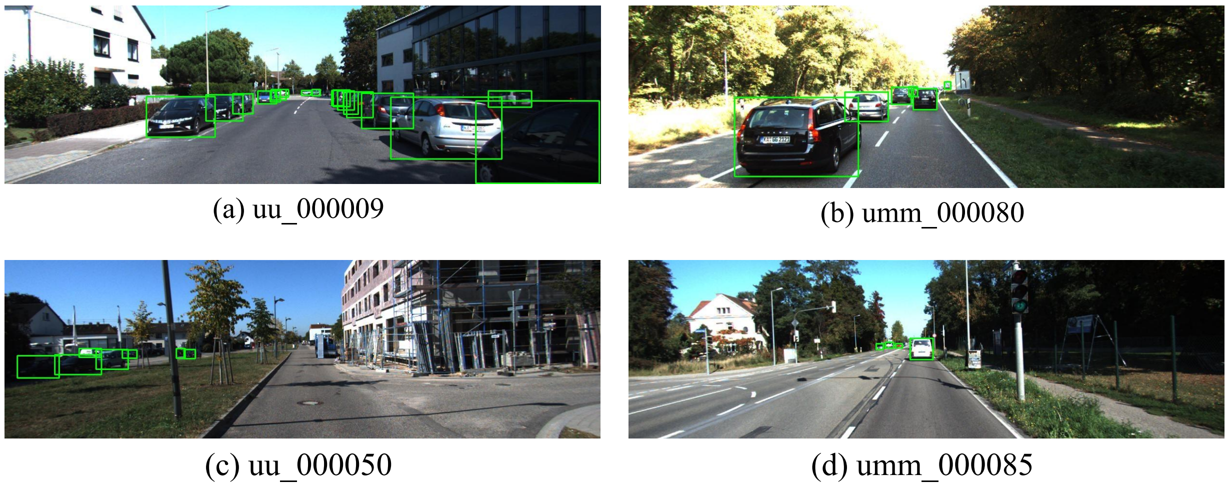


Figure 12: Faster R-CNN improvement [13]

The model implementation is available in Appendix G, and was based on an OpenCV tutorial [15]. The ***get_prediction*** method identifies all object bounding boxes and corresponding class labels, and the ***object_detection_api*** filters the car bounding boxes and marks them on the image. Figure 13 shows the results of car detection on sample test images. It is clear that Faster

R-CNN generates state-of-the-art results, as it is able to identify vehicles in a variety of different settings. Edge cases it could improve on include regions with poor lighting conditions or instances when vehicles are occluded to a significant degree.


(a) uu_000009

(b) umm_000080

(c) uu_000050

(d) umm_000085

Figure 13: Car detection results on sample test images

6 Viewpoint Classification

To predict the viewpoint of cars, feature vectors must be constructed for the image content inside the bounding boxes. These features are then passed into an MLP classifier, for the same reasons mentioned in Section 3.2. After some research, histogram of oriented gradients (HOG) descriptors [16] were selected as the feature vectors.

6.1 Histogram of Oriented Gradients (HOG) Descriptors

HOG descriptors summarize the distribution of gradient directions of an image. This is ideal for the task of viewpoint classification, where the goal is to identify the orientation or overall direction of the car. HOG descriptors are also much faster to compute than many feature extraction techniques including Scale-Invariant Feature Transforms (SIFT) [16].

The problem with HOG descriptors (as with many other feature extraction methods) is that different-sized images result in different-sized feature vectors. As a result, contents inside bounding boxes were resized to the average size of all bounding boxes (115 × 60). Alternative approaches include dimensionality reduction techniques such as Principal Component Analysis (PCA), which can be investigated in future work.

The code for extracting the HOG descriptors for each car is shown in the ***extractFeatures*** methods of Appendix H. The ***hog*** method from the Python library ***skimage.feature*** was used to generate HOG descriptors of input 115 × 60 image content [17]. The ***processAnnotations*** method in Appendix H extracts relevant information, including image content within bounding box coordinates, as well as viewpoint labels. Recall bounding box coordinates and viewpoint ground truths are provided in binary MATLAB files. For classification purposes, viewpoints are grouped by 30° increments (i.e. 12 labels from 0-11 where label *n* corresponds to the angle range *30n-30(n+1)*).

6.2 Multilayer Perceptron Classifier

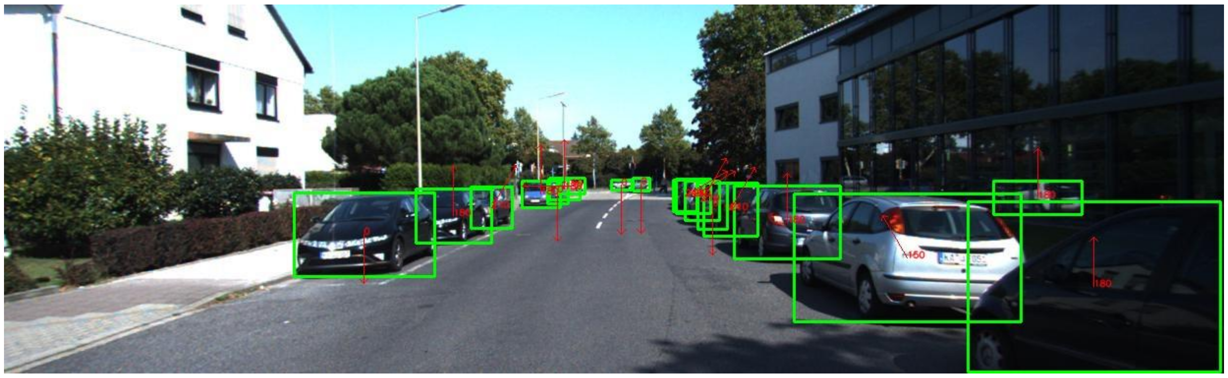
An MLP classifier was trained on the train images provided in the *train_angle* directory. A validation set was curated from the existing train set (80%-20% split) to gauge performance. Appendix I contains the code for training the classifier and computing performance metrics on the validation set. Table 3 summarizes the classification metrics on the validation set.

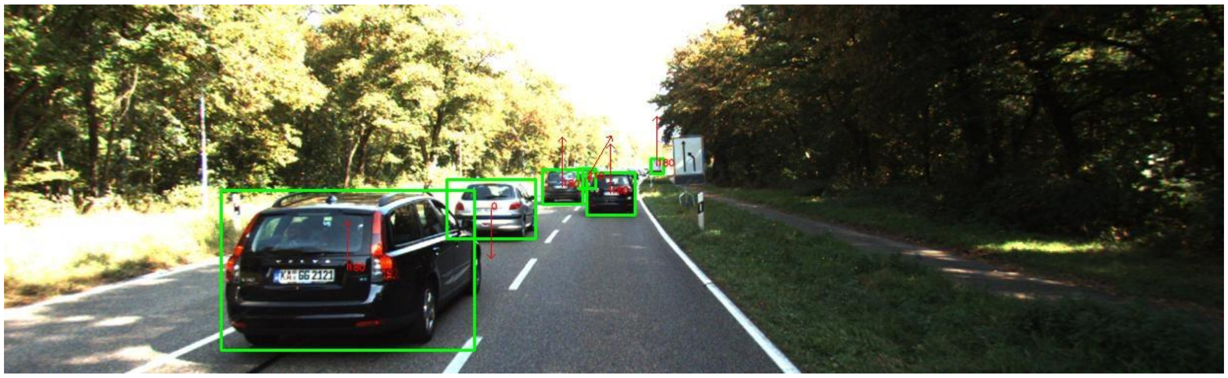
Dataset	Precision	Recall	Accuracy	Examples
Viewpoint 0	0.76	0.74	0.75	46
Viewpoint 1	0.69	0.85	0.76	34
Viewpoint 2	0.90	0.90	0.90	10
Viewpoint 3	0.64	0.70	0.67	10
Viewpoint 4	0.75	0.67	0.71	9
Viewpoint 5	0.69	0.65	0.67	17
Viewpoint 6	0.86	0.84	0.85	43
Viewpoint 7	0.57	0.67	0.62	6
Viewpoint 8	0.40	0.50	0.44	4
Viewpoint 9	0.67	0.29	0.40	7
Viewpoint 10	1.00	1.00	1.00	1
Viewpoint 11	0.00	0.00	0.00	3
All Viewpoints	**0.73**	**0.74**	**0.73**	**190**

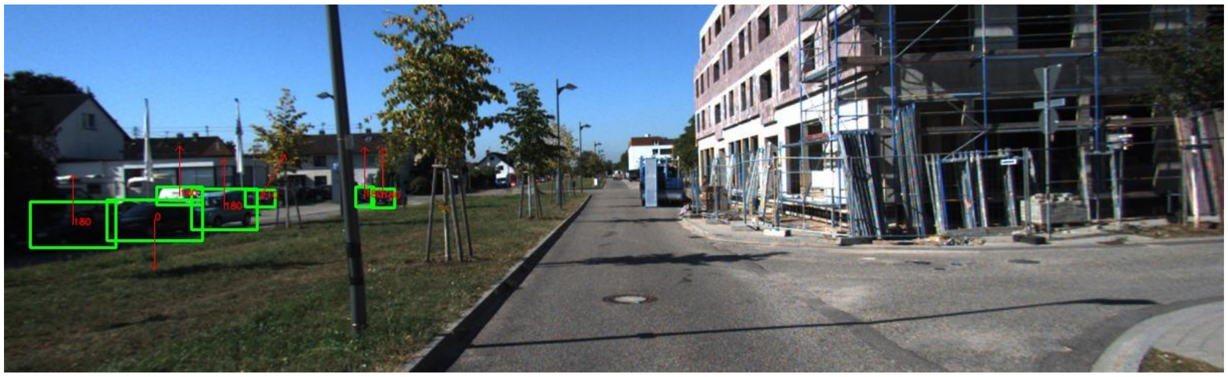
Table 3: Classification performance metrics of viewpoint classifier

The overall validation set accuracy is 73%, which is acceptable but can be improved with more complex deep learning architectures (as shown in Section 3.3). The distribution of the dataset is also skewed towards particular viewpoints (0, 1, and 6), so results for lower represented classes are unreliable (especially viewpoints 7-11).

Figure 14 visualizes the viewpoint prediction results on the test set using arrows to represent the direction. Arrows pointing straight down represent an orientation of 0°. From the results, most of the predictions are either 0° or 180° (i.e. viewpoint 0 and 6), which aligns with the distribution of the train set images. The confusion rate between these classes is observable, as cars with the same orientation have arrows pointing in opposite directions. This can be improved by adding features accounting for images of the component itself (e.g. SIFT keypoints) or using a CNN-based architecture. It also struggles with underrepresented viewpoints due to lack of available training data. This can be addressed in future iterations by looking into methods to augment data, leverage pre-trained classifiers, or assign a higher weight to these classes in the training process.


(a) uu_000009


(b) umm_000080


(c) uu_000050

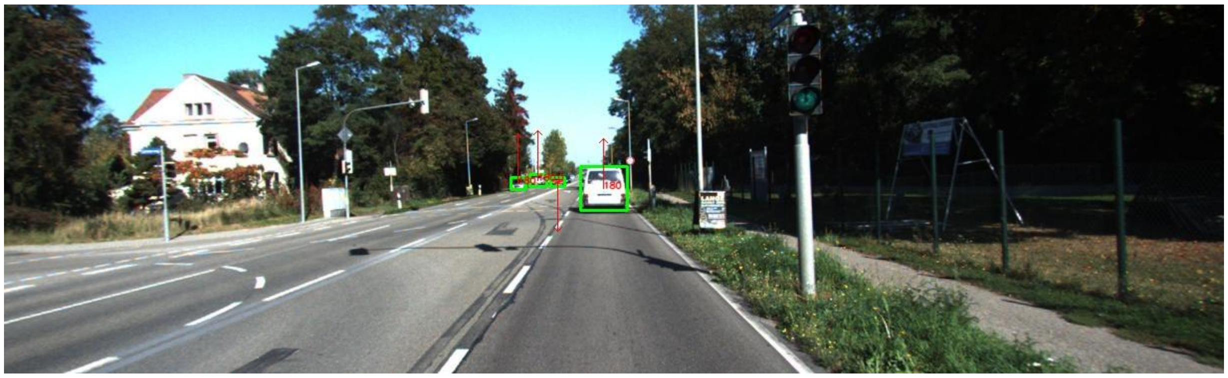

(d) umm_000085

Figure 14: Car detection and viewpoint classification results on sample test images

7 3D Bounding Boxes

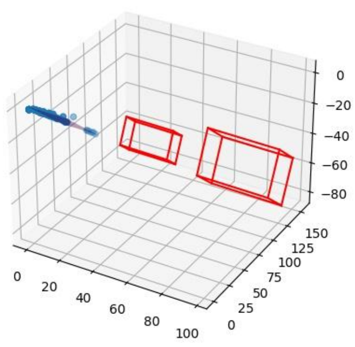
This task generates 3D bounding box approximations of a car, provided the location of a car's bounding box (in 2D), the depth map of the image, and the road plane. The step-by-step algorithm of how this is accomplished is described below.

1. Assume the pixel at the center of the 2D bounding box to also represent the center of the 3D bounding box. Find the coordinates of the 2D center (x_c, y_c), extract its depth from the image depth map (z_c) and compute the 3D center (X_c, Y_c, Z_c) using the formulas mentioned in Section 3.1.
2. Iterate through all the pixels in the bounding box and find their 3D representations. The minimum and maximum values along each dimension (i.e. X, Y, Z) will form the corners of the cube as follows: $(X_{min}, Y_{min}, Z_{min})$, $(X_{min}, Y_{min}, Z_{max})$, $(X_{min}, Y_{max}, Z_{min})$, $(X_{min}, Y_{max}, Z_{max})$, $(X_{max}, Y_{min}, Z_{min})$, $(X_{max}, Y_{min}, Z_{max})$, $(X_{max}, Y_{max}, Z_{min})$, $(X_{max}, Y_{max}, Z_{max})$. Note that only 3D

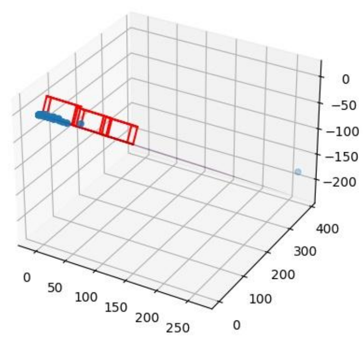
points within some Euclidean distance from the 3D center will be considered to restrict the volume of the bounding box (some occurrences of really high/low coordinates). This accounts for unreasonably high/low depth values. Through some trial-and-error, we set the threshold distance to be 30 units.

3. Rotate the 3D bounding box such that the top and bottom faces are parallel to the road plane. This is done by following the sequence of steps outlined below.
   a. Shift the 3D bounding box by ($-X_c$, $-Y_c$, $-Z_c$) such that it is centered on the origin.
   b. Compute the rotation matrix R (3×3) to be $R = [v_1\ v_2\ n]$ given the road plane $Z = aX + bY + c$ where $n = [a\ b\ -1]^T$ is the normal to the plane and v_1 and v_2 are computed as follows: $v_1 = \frac{n \times [1\ 0\ 0]^T}{\|n \times [1\ 0\ 0]^T\|}$, $v_2 = \frac{n \times v_1}{\|n \times v_1\|}$.
   c. Apply the rotation matrix to the corners of the cube by stacking the 8 corners in a 8×3 matrix P and computing the matrix product $P_R = PR$. Shift the result back by (X_c, Y_c, Z_c) to re-center the bounding box at its original position.
4. Compute the vertical distance from the bottom face of the rotated bounded box to the plane (can be done using any of the 4 bottom corners). Shift the bounding box such that the bottom face lies on the plane.

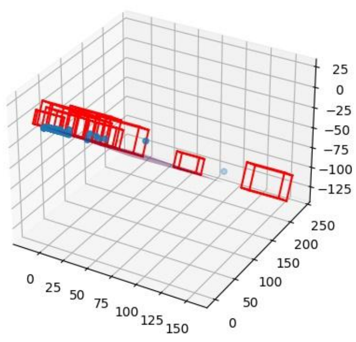
The code for the 4-step procedure can be accessed in Appendix J. Figure 15 contains visualizations of the 3D bounding boxes of some sample test images. It demonstrates a solid attempt at this problem, as all bounding boxes lie on top of the road plane. There is some overlap between bounding boxes which wasn't accounted for when initially devising the algorithm. This can be resolved with a modified threshold distance, or automatically modifying corner values once all bounding boxes have been computed. The results could also be improved by applying deep-learning models, instead of relying on simple geometry.



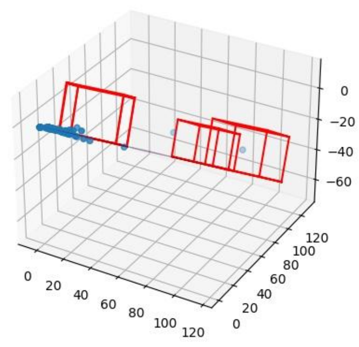
(a) uu_000012



(b) umm_000069



(c) uu_000027



(d) umm_000093

Figure 15: Estimated 3D bounding boxes (red) of cars in sample test images

8 Conclusion

In this project, a series of autonomous driving related tasks was completed. First, disparity and depth maps were computed provided a pair of left and right images captured by parallel stereo cameras. Then, different classifiers were trained (MLP classifier, U-Net based classifier) to detect road pixels; from these pixels, RANSAC was applied to estimate the road plane. After this, cars were detected from images using Faster-RCNN, and an algorithm was devised to compute the 3D bounding boxes around the car. Finally, a MLP classifier was trained to detect the viewpoint or car orientation.

In future iterations, normalization methods and dimensionality reduction techniques will be investigated to improve classification performance. The viewpoint classifier training process will also be revised to account for underrepresented viewpoint classes. Lastly, deep learning approaches can be examined to improve the 3D bounding boxes.

9 References

[1] McKinsey & Company, "Autonomous driving's future: Convenient and connected," McKinsey & Company, 17-Jul-2019. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected.

[2] E. O. Jnr, E. A. Nnabugwu, and J. D. Waziri, "Research in Autonomous Driving: A Historic Bibliometric View of the Research Development in Autonomous Driving," Research Leap, 03-Dec-2019. [Online]. Available: https://researchleap.com/research-in-autonomous-driving-a-historic-bibliometric-view-of-the-research-development-in-autonomous-driving/.

[3] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," International Journal of Robotics Research (IJRR), vol. 32, no. 11, pp. 1231-1237, Sep. 2013, doi: 10.1177/0278364913491297.

[4] H. Hirschmuller, "Stereo Processing by Semiglobal Matching and Mutual Information," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 2, pp. 328-341, Feb. 2008, doi: 10.1109/TPAMI.2007.1166

[5] "cv::StereoSGBM Class Reference," OpenCV, 2021. [Online]. Available: https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html.

[6] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2274-2282, Nov. 2012, doi: 10.1109/TPAMI.2012.120.

[7] "sklearn.neural_network.MLPClassifier," scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.

[8] "Computational Performance," scikit-learn, 2014. [Online]. Available: https://scikit-learn.org/0.15/modules/computational_performance.html.

[9] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[10] Oktay, Ozan, et al. "Attention u-net: Learning where to look for the pancreas." arXiv preprint arXiv:1804.03999, 2018. [Online]. Available: https://arxiv.org/abs/1804.03999.

[11] "sklearn.linear_model.RANSACRegressor," scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html.

[12] "Open3D: A Modern Library for 3D Data Processing," Open3D, 2021. [Online]. Available: http://www.open3d.org/.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv preprint arXiv:1506.01497, 2015. [Online]. Available: https://arxiv.org/abs/1506.01497.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.pdf.

[15] "Faster R-CNN Object Detection with PyTorch," Learn OpenCV, 2021. [Online]. Available: https://learnopencv.com/faster-r-cnn-object-detection-with-pytorch/.

[16] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005, vol. 1, pp. 886-893, doi: 10.1109/CVPR.2005.177. [Online]. Available: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf.

[17] "HOG (Histogram of Oriented Gradients)," scikit-image, 2021. [Online]. Available: https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_hog.html

10 Appendices

[A] https://github.com/esmatsahak/CSC420-Project/blob/main/depth.py

[B] https://github.com/esmatsahak/CSC420-Project/blob/main/roadFeatures.py

[C] https://github.com/esmatsahak/CSC420-Project/blob/main/viewpointClassifier.py

[D] https://github.com/esmatsahak/CSC420-Project/blob/main/road_classifier_unet.ipynb

[E] https://github.com/esmatsahak/CSC420-Project/blob/main/plane.py

[F] https://github.com/esmatsahak/CSC420-Project/blob/main/3DPointCloud.py

[G] https://github.com/esmatsahak/CSC420-Project/blob/main/objectDetection.py

[H] https://github.com/esmatsahak/CSC420-Project/blob/main/viewpointFeatures.py

[I] https://github.com/esmatsahak/CSC420-Project/blob/main/viewpointClassifier.py

[J] https://github.com/esmatsahak/CSC420-Project/blob/main/3DBoxes.py