CSC420 Project 2 - Enhancing Road Perception in Autonomous Vehicles: Integrative Approaches with Stereo Vision and Car Detection

Ian Quan
ian.quan@mail.utoronto.ca

Douglas Quan
douglas.quan@mail.utoronto.ca

Department of Mathematical and Computational Sciences
University of Toronto
Mississauga, ON, L5L 1C6

Abstract

This paper addresses critical advancements in autonomous driving technologies, emphasizing stereo image processing and deep learning-based object detection. We present a comprehensive system for depth estimation from stereo images, utilizing the Stereo Block Matching algorithm and KITTI Vision Benchmark Suite for empirical validation. Further, we explore road classification using a U-Net architecture, demonstrating enhanced segmentation accuracy through fusion techniques that integrate depth data with road annotations. Additionally, we investigate vehicle detection employing YOLOv5 and YOLOv8 models, showing significant improvements in detection accuracy and processing speed. Our results underline the potential of integrating advanced image processing and machine learning methods to improve autonomous driving systems, providing a foundation for future explorations in this rapidly evolving field.

1 Introduction

Autonomous driving represents a frontier in modern artificial intelligence applications, combining advanced computational techniques with real-world vehicle operation. Key challenges in this domain include accurate environment perception and decision-making capabilities under varying conditions. This paper explores three main areas: stereo image disparity for depth estimation, road surface classification, and vehicle detection in dynamic scenes. By leveraging the KITTI dataset, known for its complex urban driving scenarios, we aim to refine depth perception from stereo images and enhance the accuracy of object detection and road classification through state-of-the-art machine learning models. These investigations are crucial for advancing autonomous driving technologies, offering insights into effective integration of deep learning techniques with traditional image processing.

2 Stereo Image Disparity and Depth (q1, q2)

2.1 Data

The dataset employed for this project is sourced from the KITTI Vision Benchmark Suite, specifically designed for autonomous driving research. The KITTI dataset is widely recognized for its rich set of high-resolution stereo image pairs captured in urban environments. The dataset includes several categories of data, but for the purpose of this project, we focus on the stereo and calibration data, which include:

1. **Stereo Images:** Pairs of synchronized left and right images taken from a stereo camera mounted on a moving vehicle. The high resolution provides detailed pixel-level data necessary for accurate depth estimation. The stereo camera system used in KITTI consists of two camera. The cameras are mounted on the vehicle's roof with a baseline of approximately 0.54 meters between them. These stereo images are primarily used for extracting disparity and computing depth.

2. **Calibration Files:** Each stereo pair in the dataset comes with associated calibration files. These files contain intrinsic parameters of the cameras (focal length and optical centers) and extrinsic parameters detailing the relative positions and orientations of the stereo cameras. These parameters are crucial for transforming disparity values into depth measurements. The calibration file also contains information of the baseline distance, which is the distance between the centers of the two camera lenses, is used to calculate the depth from the disparity. The focal length and baseline are extracted using the provided calibration files for each image pair.

2.2 Compute Disparity Between Two Stereo Images

2.2.1 Algorithm

The algorithm employed for disparity computation between stereo images is the Stereo Block Matching (SBM) algorithm, facilitated by OpenCV's `cv2.StereoBM_create()` function. This method is a widely-used technique in stereo vision to estimate disparity maps, which represent the pixel distance between corresponding points in left and right stereo images.

Block Matching Process: The fundamental concept of SBM involves sliding a block (a small window of pixels) from the left image across a corresponding search range in the right image and finding the best match using a cost function. The disparity for each pixel is calculated based on the horizontal shift between the matching blocks that results in the minimum cost, typically measured in terms of pixel intensity differences.

2.2.2 Parameters

1. **numDisparities:** This parameter defines the maximum disparity difference that the algorithm will search. It essentially sets the range of horizontal shifts and must be divisible by 16. A higher value increases the search range but also the computational complexity.

2. **blockSize:** Represents the size of the square block (window) that the algorithm uses to compare segments of the two images. A larger block size might improve the robustness of the disparity estimation, especially in texture-less regions, but can also smooth out fine details.

To find the best combination of numDisparities and blockSize, disparity maps were computed and visualized. These experiments helped in understanding the trade-offs between computational efficiency and disparity map quality, allowing for an informed selection of parameters best suited for the given dataset. In this project numDisparites=96 and blockSize=21 were selected.

For objects closer to the camera, the distance between their projections on the left and right images increases, resulting in a larger disparity. These objects appear with higher intensity on the disparity map. Conversely, objects that are further from the camera project are closer together on the left and right images, yielding smaller disparity values with darker areas.

2.3 Depth of each pixel

Extracting depth can be done in varying forms depending on the given sensor and data. For monocular camera, directly estimating the depth of a still image is not straightforward without external aids, such as employing supervised deep learning models or utilizing parallax techniques. In contrast, stereo vision cameras provide another method for depth extraction.

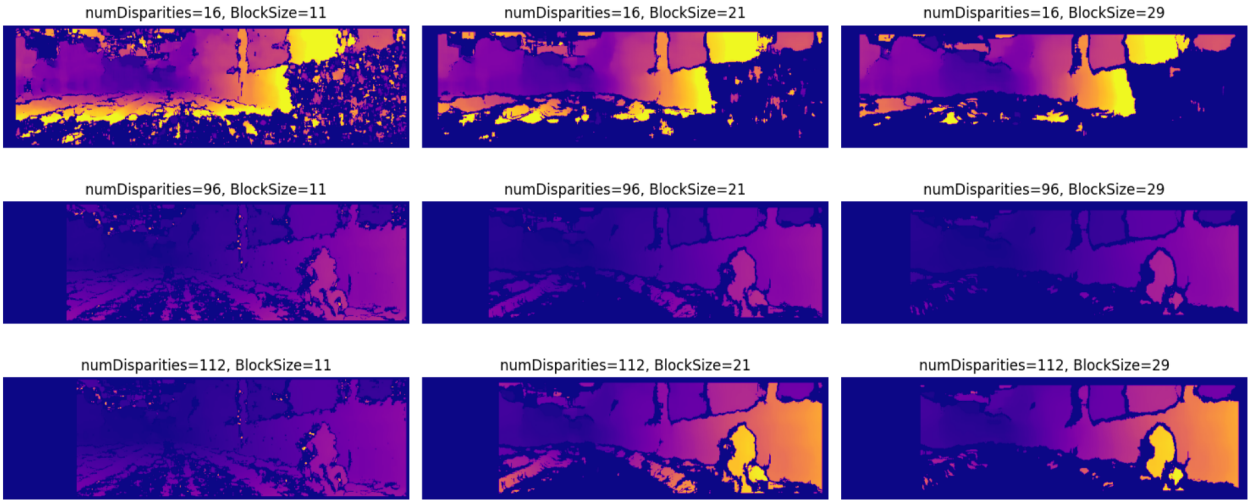

Figure 1: Disparity Map with different block size and maximum disparity difference

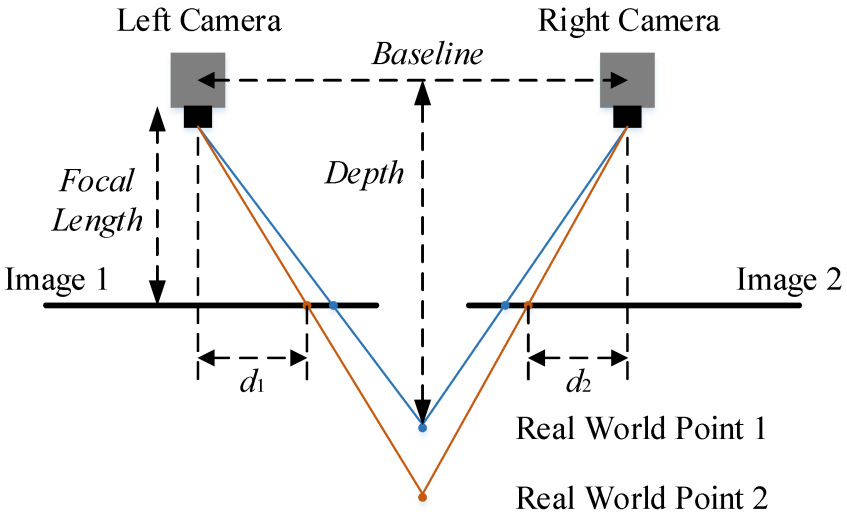

Figure 2: A basic model of stereo cameras with intrinsic camera parameters, from Liu et al. [2016]

2.3.1 Calculation of depth

As depicted in Figure 2, stereo vision involves projecting a point in 3D space onto two different frames, which appear differently due to the cameras' disparate positions. Knowing the relative positions of these points allows for the calculation of the depth to the real-world point. The depth is calculated using the geometry of stereo vision. The formula to compute the depth Z of each pixel in a stereo image pair is given by:

$$Z = \frac{f \times b}{d} \tag{1}$$

where f is the focal length of the camera (in pixels), b is the baseline distance between the two cameras (in meters), d is the disparity (in pixels) at the pixel of interest.

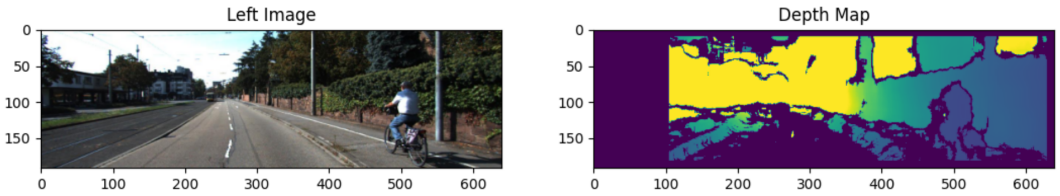

Figure 3: Original image to the left and depth map to the right. Pixels with higher intensity indicates a larger depth and vice versa

3 Road Classification (q3)

3.1 Data Processing

For this project, the KITTI Vision Benchmark Suite is employed, specifically utilizing a subset of its comprehensive data collections tailored for autonomous driving applications. The dataset consists of 135 stereo image pairs, each accompanied by ground truth annotations. High-resolution stereo images are essential for extracting depth information and understanding the spatial layout of the surroundings.

3.1.1 Pre-processing the Data and Data Splitting

The initial step in preparing the dataset for the road classification task involves several key processes aimed at optimizing the input data for effective model training. The pre-processing routine includes:

Normalization: All image inputs and depth maps are normalized to a range between 0 and 1. This step is critical as it facilitates faster convergence during training and maintains numerical stability.

Mask Normalization: The ground truth masks, which identify road areas, are converted to a floating-point format and normalized, ensuring that mask values are either 0 or 1. This binary format directly corresponds to the presence or absence of the road.

After pre-processing, the dataset is split into training, validation, and testing sets. Using an 80-20 split, 80% of the data is reserved for training and validation, while the remaining 20% is set aside for final testing. This separation helps in evaluating the model on unseen data, providing a measure of its generalization capability. Further, the training-validation portion undergoes another split where 75% is used for training and 25% for validation purposes. This approach ensures that the model is not only learning effectively but also tuning its parameters to avoid overfitting.

3.1.2 Fusing Depth Map and Road Ground Truth

To enhance the model's ability to discern road surfaces from other objects and terrain, depth maps and road masks are fused. This fusion process involves:

Depth Inversion: The normalized depth map is inverted (1 - depth value), placing nearer objects (typically higher in depth) lower in intensity which aids in distinguishing them from the road surface.

Fuse Inverted Depth Map With Ground Truth: To create a new fused ground truth, matrix multiplication was performed between the depth map and original ground truth. As shown in Equation 2. Considering that the original ground truth only contains the values 0, or 1, i.e. background or road, the zero background values will eliminate (set each pixel to zero) the irrelevant depth. While the road mask (pixel value of 1) will yield that all the road pixels for each image retain their depth values. The fusion ensures that depth information is only retained for the road areas, significantly reducing background noise and focusing the model's learning on road features.

$$a_{ij} \cdot b_{ij} = \begin{pmatrix} a_{11} \cdot b_{11} & \cdots & a_{1n} \cdot b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} \cdot b_{m1} & \cdots & a_{mn} \cdot b_{mn} \end{pmatrix}, \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, m \tag{2}$$

a_{ij} is the ground truth with ones and zeros, and b_{ij} corresponds to the inverted depth map

This fused depth map serves as the target output (labels) for the training process. An illustration of the fused depth map from an example image can be seen in Figure 4

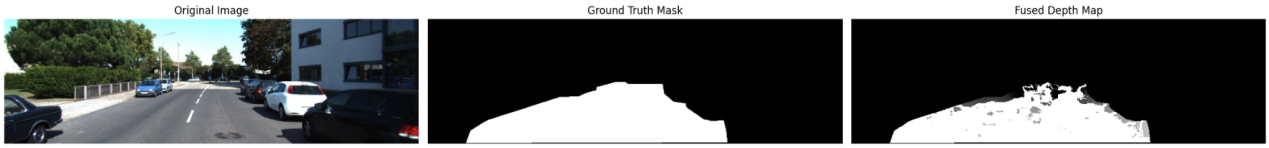

Figure 4: Fused ground truth with depth information

3.1.3 Data Augmentation

Given the limited size of the dataset, data augmentation is particularly crucial. It effectively enlarges the training dataset, providing the model with a broader range of scenarios and features to learn from.

Data augmentation artificially add variation to the dataset by introducing random, realistic transformations. For this project, horizontal flipping augmentation is applied. Images and their corresponding masks are flipped horizontally, simulating the reversal of driving lanes, which is particularly useful for ensuring the model's effectiveness across different driving contexts.

Both the image and mask data are augmented in sync using the same random seed to ensure that the transformations correspond exactly, preserving the integrity of the ground truth data. The augmented data is generated in batches, using a predefined batch size to optimize memory usage and computational efficiency during training. This yielded a less risk of overfitting and better generalization of the model while extending the data with varying training images.

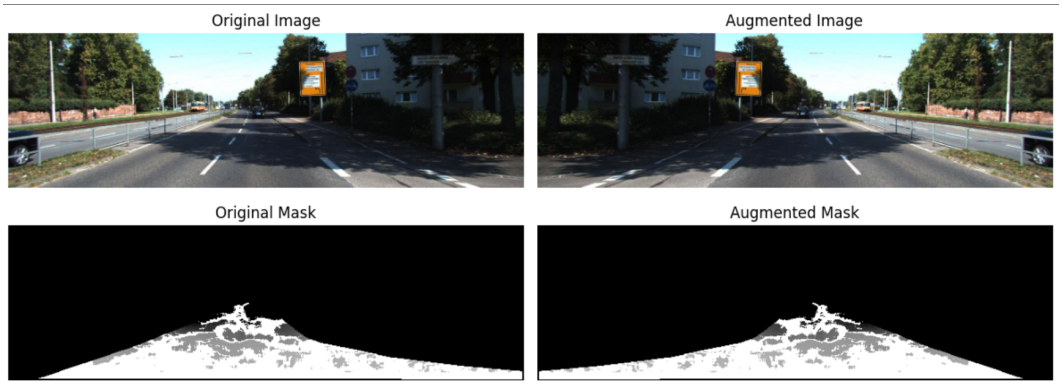

Figure 5: Example image and mask that is flipped for data augmentation

3.2 U-Net

3.2.1 Model Architecture

The U-Net architecture is a convolutional neural network originally designed for biomedical image segmentation. Its structure is particularly well-suited for tasks requiring precise localization, making it ideal for road segmentation in stereo images. The architecture can be divided into two main components: the encoder (contracting path) and the decoder (expansive path), connected by a bridge.

Encoder: The encoder consists of a series of blocks, each containing two convolutional layers followed by a batch normalization layer and a ReLU activation. These blocks are designed to capture the underlying features of the image at various resolutions and contexts. After each convolutional block, a max-pooling operation is performed, reducing the spatial dimensions by half and incrementally increasing the depth of feature maps. This process helps the network focus on the most salient features, enhancing its ability to generalize across different visual scenarios.

Decoder: The decoder mirrors the encoder's structure but inversely; it gradually upscales the feature maps to reconstruct the original image dimensions. Each step in the decoder begins with a transposed convolution, effectively doubling the size of the feature maps. These are then concatenated with the corresponding encoded feature maps preserved from the encoder through skip connections. These connections are crucial as they reintroduce the spatial details lost during downscaling, allowing for precise localization and detailed segmentation.

Output Layer: The final layer of the network is a convolutional layer with a sigmoid activation function, which maps the final feature representation to the desired output format. In the case of road segmentation, the output is a binary mask that classifies each pixel as road or non-road.

The U-Net architecture can be visualized as a U-shaped structure, where the bottom of the "U" represents the bridge, and the two arms are the encoder and decoder. This design facilitates the flow and refinement of feature maps throughout the network, ensuring detailed and accurate segmentation outputs. With the U-Net, we can solve the two questions of segmentation: "what" and "where."

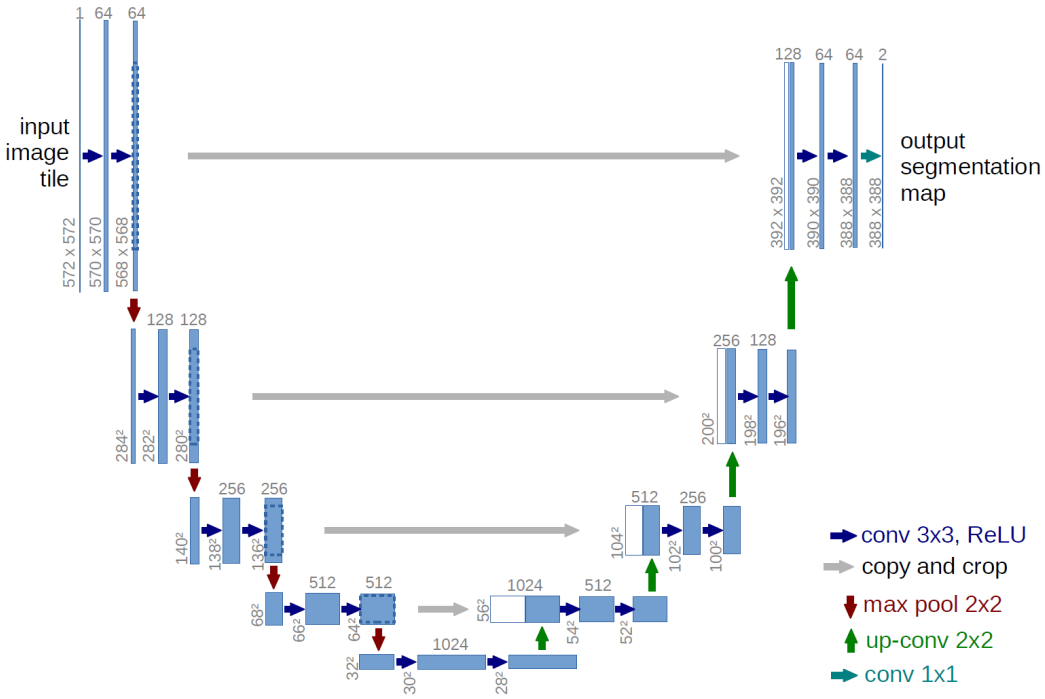

Figure 6: U-Net Architecture from Ronneberger et al. [2015]

3.2.2 Rationale Behind the Choice of using U-Net for Road Semantic Segmentation

This decision is driven by several key factors that make U-Net particularly suitable for this application:

High Precision Localization: U-Net excels in tasks where precise localization is crucial. For road segmentation, accurately delineating the boundaries of the road is important for safe navigation of autonomous vehicles. U-Net's architecture, with its contracting and expansive paths, ensures that both high-level context and fine-grained detail are captured and utilized effectively.

Efficiency with Limited Data: Originally developed for medical image segmentation where datasets are typically small, U-Net is designed to perform well with limited data through its use of extensive data augmentation and efficient use of image data via skip connections. This capability is particularly advantageous given the limited size of the KITTI dataset used in our project.

Skip Connections: These are integral to U-Net and allow the transfer of fine-grained details from the encoder to the decoder network. In road segmentation, these connections help preserve important spatial information (like edge details of the road) that might be lost during the pooling operations in the encoder. By reintegrating this information during the decoding phase, U-Net ensures more accurate and detailed segmentation results.

End-to-End Learning: Unlike systems that rely on hand-engineered features or multiple stages of processing, U-Net provides a straight-forward, end-to-end learning solution that directly outputs the segmentation mask from the raw input images. This simplifies the modeling pipeline and potentially reduces the time to deployment.

3.2.3 Model Evaluation

In assessing the performance of the U-Net model for road segmentation, some evaluation metrics are chosen to measure the performance of the neural networks. The metrics chosen are among several that are the most suitable for this evaluation purpose, used by many state of the art research paper within the same field. The primary metrics used for evaluating our U-Net model include:

Dice Coefficient: This metric, also known as the Sørensen–Dice coefficient, is a statistical tool that measures the similarity between two samples. It is particularly well-suited for evaluating models on segmentation tasks because it compares the pixel-wise agreement between the predicted segmentation masks and the true masks. The Dice coefficient is defined as:

$$\text{Dice Coefficient}(\hat{y}, y) = \frac{2\text{TP}(\hat{y}_i, y_i)}{2\text{TP}(\hat{y}_i, y_i) + \text{FP}(\hat{y}_i, y_i) + \text{FN}(\hat{y}_i, y_i)} \tag{3}$$

where $\hat{y}_i$ is the predicted value of the pixel and y_i being the ground truth value of the labeled pixel i

Intersection over Union (IoU): Also known as the Jaccard index, this metric measures the overlap between the predicted mask and the true mask. The IoU is calculated by dividing the area of overlap between the predicted and actual masks by the area of union between them. Like the Dice coefficient, a higher IoU score indicates better segmentation accuracy. This metric is especially critical for tasks where boundaries and precise delineation are important, such as road segmentation. The Intersection over Union is defined as:

$$\text{IoU}(\hat{y}, y) = \frac{\text{TP}(\hat{y}_i, y_i)}{\text{TP}(\hat{y}_i, y_i) + \text{FP}(\hat{y}_i, y_i) + \text{FN}(\hat{y}_i, y_i)} \tag{4}$$

where $\hat{y}_i$ is the predicted value of the pixel and y_i being the ground truth value of the labeled pixel i

Accuracy: This is a straightforward metric that measures the proportion of correctly identified pixels (both road and non-road) in the segmentation mask. While accuracy is a useful metric, it can sometimes be misleading, especially in cases where there is a significant class imbalance (e.g., much more non-road area than road). That is why our model's loss function is based on the negative Dice coefficient instead of accuracy.

$$\text{Accuracy}(\hat{y}, y) = \frac{\text{Matched Predictions}}{N}, \quad \text{Matched Predictions} = \sum_{i=1}^{N} \begin{cases} 1, & \hat{y}_i = y_i \\ 0, & \hat{y}_i \neq y_i \end{cases} \tag{5}$$

Choice of Dice Coefficient for Loss Function

The decision to use the Dice coefficient for the loss function, rather than solely relying on accuracy, stems from its effectiveness in handling the class imbalance typically found in road segmentation

tasks. The Dice loss (defined as 1 - Dice Coefficient) helps to directly optimize the model towards increasing the overlap between the predicted and ground truth masks, ensuring that both the presence and precise localization of roads are accurately captured. The choice of Dice coefficient for the loss function promotes better generalization across varying scenes and lighting conditions, ensuring the model remains robust and reliable under different operational environments.

3.2.4 Model Training and Model Parameters

Training Process

The training process of a U-Net model begins with initializing the U-Net architecture, compiling it with the Adam optimizer and Dice coefficient loss. Throughout the training, the model monitors several key metrics: traditional accuracy, the Dice coefficient itself, and the Intersection over Union (IoU). These metrics provide a comprehensive view of the model's performance, emphasizing both general accuracy and the specific quality of the segmentation. To prevent overfitting and enhance training efficiency, an early stopping callback is implemented, which halts training if there is no improvement in the validation Dice coefficient for five consecutive epochs.

Model Parameters

The key parameters of the U-Net model include:

1. **Input Shape:** The input resolution, as well as the output resolution for all images were 640×192. The input was however a RGB image while the output was in gray-scale as mentioned.

2. **Optimizer:** ADAM optimizer with a initial learning rate of 0.001.

3. **Loss Function:** Negative Dice coefficient, optimizing for overlap between predicted and true segmentation.

4. **Batch Size:** Set to 8 to manage computational resources effectively while allowing for sufficient gradient approximation.

5. **Epochs:** 100, with early stopping based on validation performance to prevent overfitting. The number of epochs is larger than normal due to the lack of training data.

3.3 Results of Road Classification

The performance of the U-Net model for road segmentation was evaluated using key metrics: Pixel-Accuracy, Intersection over Union (IoU), and Dice Coefficient. We compared the results of a standard U-Net model with a variant where a fused mask incorporating 3D features was used. The detail plots of the three metrics can be viewed in the appendix on table 20, table 21 and table 22.

From table 1, it clearly shows that the U-Net model with the fused mask significantly outperforms the standard U-Net model across all metrics. The improvements observed with the addition of the fused depth mask, underline the benefit of integrating additional contextual and depth-related information into the segmentation process. The use of 3D features allows the model to better understand the spatial relationships and structures within the images, leading to more accurate segmentation results. This approach is particularly effective for complex environments where depth cues play a critical role in distinguishing between different elements within the scene.

Table 1: Road Classification Results

Model	Pixel-Accuracy	IoU	Dice-Coefficient
U-Net	0.9486	0.8127	0.8965
U-Net with fused mask	0.9611	0.8366	0.9099

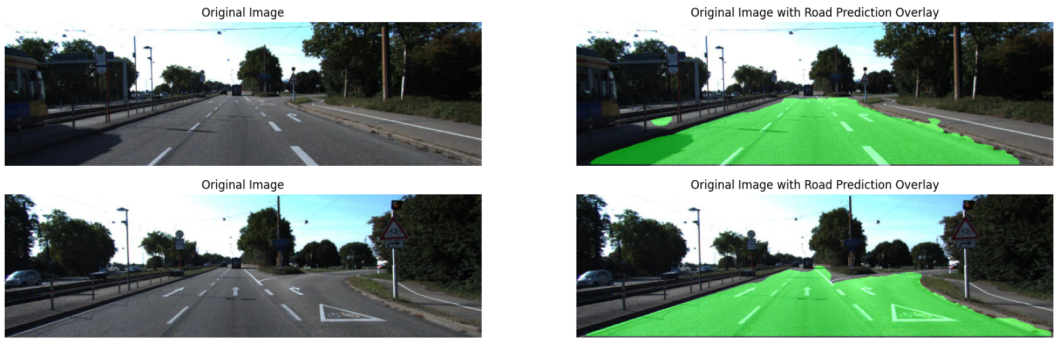

Figure 7: U-Net model result

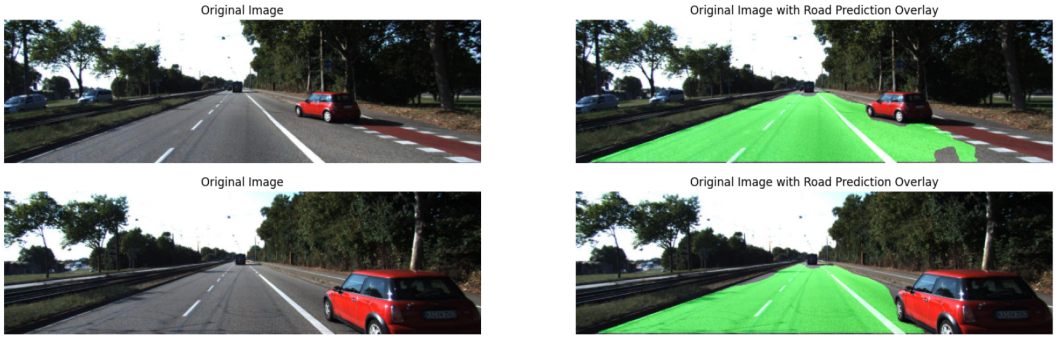

Figure 8: U-Net model result

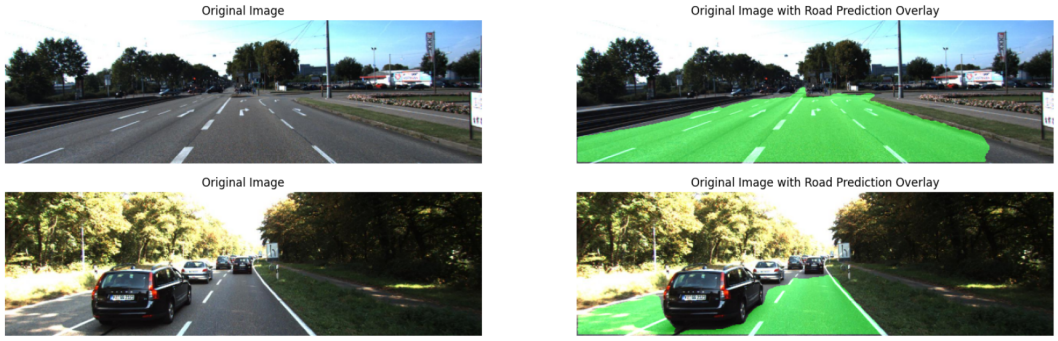

Figure 9: U-Net model result

3.4 Challenges and Limitaions

While the U-Net model has demonstrated considerable success in segmenting road areas from stereo images, there are inherent challenges and limitations that affect its performance. These challenges are particularly evident in complex traffic scenarios and under specific road conditions.

3.4.1 Classification in the Presence of Lane Markings

One of the significant challenges encountered is the model's performance around lane markings, especially when there are distinct lines separating lanes. The presence of these lines can confuse the

model, leading to inaccurate classifications where parts of the road are misclassified as non-road areas. This issue stems from the model's sensitivity to high-contrast features, which can overpower the more subtle cues that define continuous road surfaces.

Solution: Expand the dataset to include a wider variety of road conditions, including various types of lane markings and more diverse traffic scenarios. More extensive data collection efforts could help in training models that are more adaptive and robust to the complexities of real-world driving.

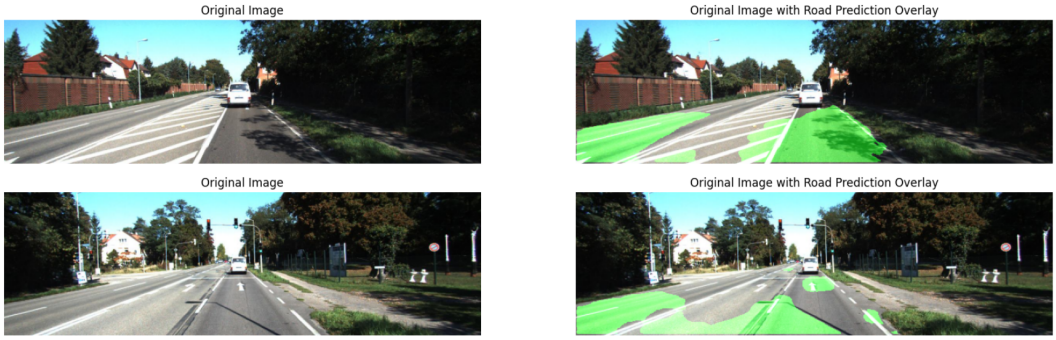

Figure 10: Example of misclassified road pixels in lane marking condition

3.4.2 Insufficient Training Data

Another critical limitation is the amount of training data available. The model's ability to generalize well across various road conditions heavily relies on the diversity and volume of data used during training. Currently, the dataset, while robust in many respects, does not comprehensively cover all possible variations in road conditions, weather scenarios, and traffic densities. Limited data particularly affects the model's performance in less common but crucial scenarios, such as roads with atypical lane markings, temporary road signs, or unusual weather conditions. This limitation can lead to a lack of robustness in the model, where it performs well under normal conditions but fails under less typical or more challenging conditions.

4 Plane Fitting for Road Surface in 3D point Cloud(q4, q5)

4.1 Methodology

Fitting a plane to the road surface within a 3D point cloud comprised several key steps:

4.1.1 3D Coordinate Extraction

For each pixel identified as part of the road from the output of the road classification model, its 3D coordinates were calculated using the depth map generated from stereo images using equation 1. The depth map provides the distance from the camera to each pixel, while the camera's intrinsic parameters (focal length and principal points) allow for the conversion of these depth values into 3D coordinates (X, Y, Z).

Given:

- Z: Depth of the pixel from the camera, obtained from the depth map with equation 1
- p_x, p_y: Principal points of the camera, obtained from the calibration files.
- f: Focal length of the camera.
- u, v: Original pixel coordinates in the image's 2D space.

The 3D coordinates (X, Y, Z) are computed as follows:

$$X = \frac{(u - p_x) \times Z}{f} \tag{6}$$

Here, $(u - p_x)$ calculates the horizontal displacement of the pixel from the principal point along the x-axis, normalized by the focal length f. This value is then scaled by the depth Z to convert it into a real-world horizontal distance from the center of the camera.

$$Y = \frac{(v - p_y) \times Z}{f} \tag{7}$$

Similarly, this applies to the vertical plane.

With equation 6 and 7, the coordinates can be compactly expressed in vector form as:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = Z \times \begin{bmatrix} \frac{(u-p_x)}{f} \\ \frac{(v-p_y)}{f} \\ 1 \end{bmatrix} \tag{8}$$

For this project, the principal points p_x, p_y, focal length and baseline f can be extracted from the calibration data of the cameras used in the KITTI dataset. The values are as follows:

- Principal Point X: 609.5593
- Principal Point Y: 172.8540
- Focal Length: 721.5377 pixels
- Baseline: 0.5372 meters

4.1.2 Robust Plane Fitting:

To accurately model the road surface and be robust against outliers (which are common in real-world data due to various factors like shadows, occlusions, and non-uniform road textures), a RANSAC (Random Sample Consensus) algorithm was employed. This algorithm iteratively fits multiple models to subsets of the data, estimating the parameters of a plane that best fits the road points while ignoring outliers.

4.1.3 Visualization:

Point Cloud Visualization: The 3D coordinates of road pixels were plotted as a point cloud to provide a visual context of the data structure and distribution. Plane Visualization: Alongside the point cloud, the plane estimated by the RANSAC algorithm was plotted. This involved creating a grid of (X, Y) coordinates that span the extent of the observed data and using the plane model to predict the Z values, forming a visual representation of the estimated road plane.

4.2 Visualization

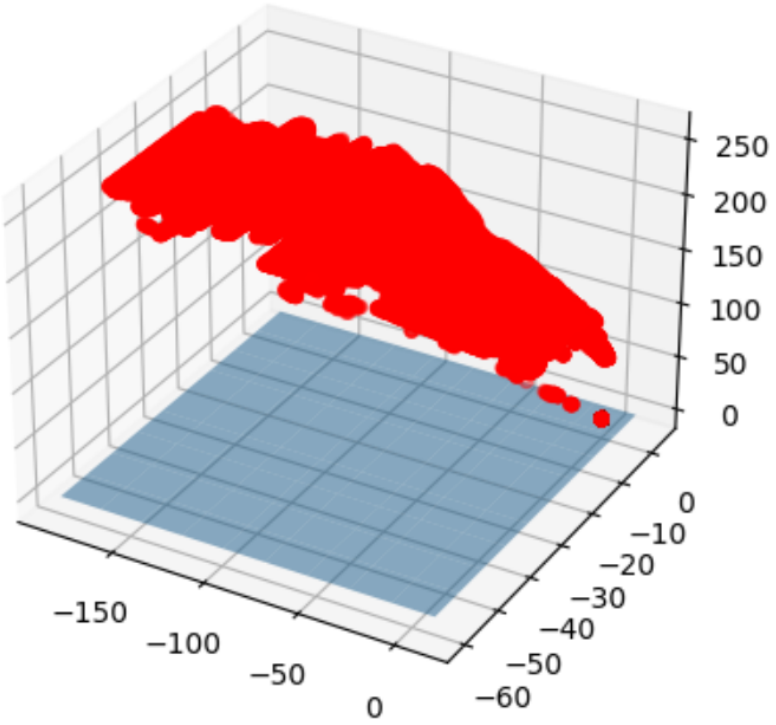

Figure 11: 3D point cloud of the road pixels with the estimated ground plane

5 Car Detection (q6)

5.1 Introduction

Object detection is a key technology in computer vision with applications ranging from surveillance to autonomous driving. Accurate and real-time detection of vehicles in images is crucial for these applications. YOLO models are among the most popular choices for such tasks due to their speed and accuracy.

5.2 Methodology

Dataset Collection: We utilized the KITTI dataset (Geiger et al. [2012]), renowned for its rich collection of images captured from vehicles equipped with various sensors, including high-resolution cameras. The dataset provides a wide range of annotated images of street scenes, making it ideal for training and evaluating object detection models focused on automotive applications.

Preprocessing: Prior to detection, images from the KITTI dataset were preprocessed to match the input requirements of the YOLO models. This involved resizing images to the models' input dimensions and normalizing pixel values. Such standardization ensures that the models' pre-trained weights are compatible with the input data.

Model Selection: For our detection task, we selected two versions of the YOLO model: YOLOv5 (Jocher [2020]) and YOLOv8 (Jocher et al. [2023]). These models were chosen based on their proven track record in object detection tasks, with YOLOv5 being known for its speed and efficiency and YOLOv8 for its improved accuracy and advanced features.

Model Loading: The pre-trained YOLO models were loaded from their respective sources. YOLOv5 is sourced from a PyTorch hub, while YOLOv8 is obtained directly from Ultralytics. Both models are well-documented and supported, ensuring ease of integration into our workflow.

Inference: With the models loaded and the images preprocessed, we performed inference on the KITTI dataset. The models processed each image, detecting objects classified as cars and generating corresponding bounding boxes with confidence scores.

Detection and Post-processing: Following inference, we applied post-processing techniques to refine the detection results. This included confidence score thresholding to filter out weak detections and non-maximum suppression to eliminate overlapping bounding boxes, ensuring each car was detected only once.

Results Display: The final detections were visualized by superimposing the bounding boxes onto the original images. Each box was accompanied by a label indicating the 'Car' class and the model's confidence in its prediction.

5.3 Rationale for Using YOLO

YOLO's unified architecture makes it extremely fast, which is a key requirement for real-time applications. Furthermore, YOLO's spatial reasoning can implicitly encode the context around objects, making its detections robust and accurate.

5.3.1 Comparison with SSD and Faster R-CNN:

Speed and Real-time Performance:
YOLO models are renowned for their speed, which is crucial for real-time applications. While SSD and Faster R-CNN are effective object detection models, YOLO generally provides faster inference times, making it better suited for applications that require real-time processing without significantly compromising accuracy. (Ste [2023])

Simplicity in Deployment:
YOLO's architecture is less complex when it comes to deployment in various environments, which can be a deciding factor for projects with limited customization or integration time.

5.3.2 Avoiding Older Models like LSVM from KITTI and DPMs:

Model Age and Performance:
The LSVM (latent SVM) model and DPMs (Deformable Part Models) are older object detection technologies. While historically significant, they do not perform as well in terms of speed or accuracy compared to newer deep learning-based models like YOLO (Wu et al. [2019]). The evolution in hardware capabilities and algorithmic innovations has shifted preference towards neural networks which outperform these older methods, especially on complex tasks like car detection in diverse scenarios.

Dataset Limitations:
Although the KITTI dataset is an excellent resource for automotive contexts, models pre-trained on KITTI using older technologies like LSVM might not generalize well outside of the specific scenarios captured in the dataset. Modern deep learning models are often trained on more diverse datasets and can be fine-tuned with transfer learning techniques to adapt to specific environments better.

5.3.3 Technological Advancements:

Advances in Deep Learning:
Since the release of models like LSVM and DPM, deep learning has made significant advances. Modern architectures like YOLO leverage these advancements to provide better feature extraction capabilities and improve detection accuracies through techniques such as end-to-end training, real-time processing speeds, and better generalization across different lighting and environmental conditions.

5.3.4 Robustness and Accuracy:

Spatial Reasoning and Context Awareness:
YOLO models not only detect objects within an image but do so by understanding the spatial hierarchy

and context, which leads to fewer false positives compared to older models like DPMs (Redmon et al. [2016]). This capability is critical in dynamic environments such as driving, where the context and object relations play a crucial role in accurate detection.

5.4 Comparison between YOLOv5 and YOLOv8:

The selection of YOLOv5 and YOLOv8 for object detection tasks is underpinned by their performance metrics and the improvements they've introduced over time. YOLOv5, released by Ultralytics in 2020, is known for its speed and efficiency, making it a popular choice for many developers due to its ease of use and state-of-the-art results for object detection tasks. YOLOv8, on the other hand, is the latest model introduced in 2023, also by Ultralytics, and is built on the YOLOv5 framework. It incorporates several architectural enhancements and provides a unified framework for various tasks like object detection, instance segmentation, and image classification. It is considered faster and more accurate than YOLOv5, which is beneficial for applications that demand real-time object detection.

In a benchmark performance test conducted on NVIDIA's Jetson AGX Orin, YOLOv8 models demonstrated a significant increase in mAP (mean Average Precision) compared to YOLOv5, indicating an improvement in accuracy while maintaining similar runtimes. Particularly, the lightweight YOLOv8 models (such as v8n) showcased the best combination of accuracy and speed, making them a strong candidate for applications that prioritize both aspects. (Ste [2023])

Table 2: results for all YOLOv8s vs YOLOv5s models in 640 resolution on both NVIDIA Jetson AGX Orin (JP5) and RTX 4070 Ti (Batch 1, TRT8.4, FP16), from Ste [2023]

Model	AP	AP0$\dot{5}$	AGX ORIN (FPS)	RTX 4070 TI (FPS)
YOLO v5s	37.4	56.8	277	877
YOLO v8s	44.9	61.8	260	925

While comparing YOLOv5 and YOLOv8, it's crucial to consider factors such as speed, accuracy, ease of use, and developer experience. YOLOv8 outperforms YOLOv5 in terms of speed and accuracy due to the improvements made in its architecture. However, for developers who prioritize ease of use and a comprehensive toolset for a broader range of tasks, YOLOv8's unified framework may be more appealing (Mas [2023])

5.5 Model Architecture

YOLOv5 leverages a streamlined and efficient design for real-time object detection. The architecture can be broken down into three main components:

Backbone:
This is the feature extraction component, which utilizes a CSP (Cross Stage Partial) network to extract feature maps at various scales. The CSP design helps in reducing computational cost while maintaining rich feature extraction capabilities.

Neck:
The neck employs a Feature Pyramid Network (FPN) combined with a Path Aggregation Network (PAN) for feature fusion. This design allows for the integration of features at different levels, enhancing the detection of objects at varying scales. The feature maps generated here are typically at three different resolutions (P3, P4, P5), corresponding to different object sizes (small, medium, and large).

Head:
The detection head uses these feature maps to perform object classification and bounding-box regression. Each pixel in the feature map contributes to the prediction of bounding boxes, leveraging preset anchor boxes to guide the predictions.

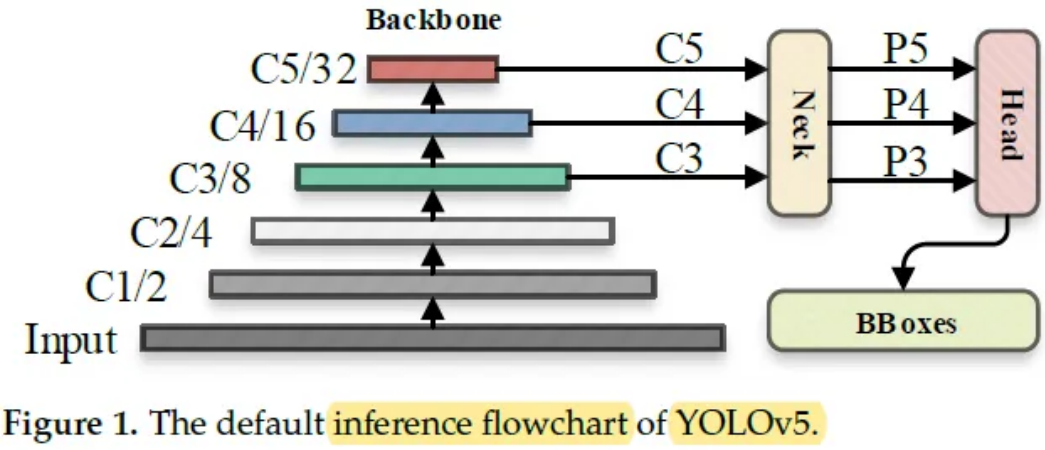

Figure 1. The default inference flowchart of YOLOv5.

Figure 12: YOLOv5: Overall Architecture, from Liu et al. [2022]

YOLOv8 introduces several enhancements over YOLOv5, aiming to improve both the precision of detections and the model's efficiency. Key changes include:

Modified Backbone:
The backbone sees a shift from the use of multiple 6x6 convolutions to simpler 3x3 convolutions, aiming to streamline computations without compromising the depth of feature analysis.

C2f Module: Replacing the C3 module with the C2f module, YOLOv8 optimizes feature processing to enhance feature extraction and integration, providing a more refined understanding of the image context.

Simplified Convolutional Layers:
By removing certain convolutional layers (specifically the 10th and 14th in the typical YOLOv5 setup), the model becomes lighter, which can speed up the inference time while maintaining adequate detection capabilities.

Bottleneck Enhancements:
The first 1x1 convolution in the bottleneck is replaced with a more effective 3x3 convolution, allowing for better feature manipulation and efficiency in the deeper layers of the model.

Decoupled Head:
This significant architectural change involves removing the objectness branch and using a decoupled approach to handle different tasks within the detection process, leading to improvements in how precisely the model can localize and classify objects.

5.6 Results

Both models demonstrated high accuracy in car detection. YOLOv8 exhibited a slight edge in precision over YOLOv5. YOLOv5 outperformed YOLOv8 in terms of speed, making it a preferable choice for time

Figure 13: YOLOv5 vs YOLOv8: result comparison

5.7 Challenges and Solutions

A primary challenge was adjusting the input size and normalization for each model without losing relevant information. This was addressed by carefully resizing images and applying the same normalization as used in model training. Another challenge involved managing false positives, which were reduced by adjusting the confidence threshold and applying non-max suppression more rigorously.

5.8 Conclusion:

In conclusion, both YOLOv5 and YOLOv8 are capable of high-performance car detection. The choice between them would depend on the specific requirements of the application in terms of speed and accuracy. YOLOv5 stands out for its fast processing times, whereas YOLOv8 is a strong candidate when the highest accuracy is required.

6 Car viewpoint classification (q7)

6.1 Methodology

6.1.1 Transfer Learning

Transfer learning has revolutionized the field of machine learning by enabling models developed for one task to be reused as the starting point for another task. This approach is particularly beneficial when data is scarce or when computational resources are limited. Pre-trained models, such as ResNet50, InceptionV3, and VGG16, have shown significant success across various domains, particularly in image recognition tasks. They are trained on large datasets like ImageNet, allowing them to develop robust feature representations for a wide range of images (Pan and Yang [2010]).

6.1.2 Rationale for Using Transfer Learning in Car Viewpoint Classification

For the task of classifying car viewpoints, using transfer learning is advantageous for several reasons:

1. Efficiency and Speed:
   Transfer learning allows for the rapid development of models with fewer data and less computational expense compared to training from scratch (Pan and Yang [2010]).
2. Proven Effectiveness:
   Studies have demonstrated that transfer learning often outperforms models trained from scratch, especially when the available training data is limited (Pan and Yang [2010]).
3. Feature Extraction Capabilities:
   Unlike traditional machine learning techniques that require manual feature extraction, Convolutional Neural Networks (CNNs) like ResNet50, InceptionV3, and VGG16 automatically learn to extract features. This automatic feature extraction is crucial for handling complex image data and contributes to the robustness of the model (Simonyan and Zisserman [2015a]).

6.1.3 Implementation Steps

1. Preparation of the Dataset:
   The initial stage of the implementation involves preprocessing the dataset to prepare it for input into the neural network models. The dataset used is a subset of the KITTI object detection and orientation estimation benchmark, which contains a total of 582 examples across 12 classes.

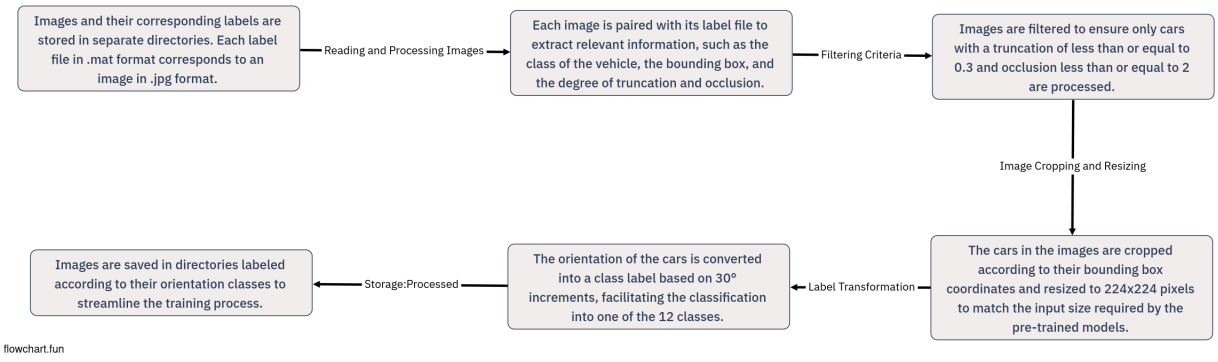

Figure 14: Workflow of Data Preparation, dataset from Geiger et al. [2012]

2. Model Training:
   With the data prepared, the next step focuses on configuring and training the models using TensorFlow and Keras. The models trained include ResNet50, InceptionV3, and VGG16, utilizing their architectures pre-trained on ImageNet. The pre-trained models were employed

as feature extractors by utilizing their convolutional bases to process car images, followed by the addition of a custom classification layer specifically designed for the 12-class viewpoint classification, and finally, the models were fine-tuned by training the newly added layers while earlier layers remained frozen to optimize performance on the car viewpoint dataset.

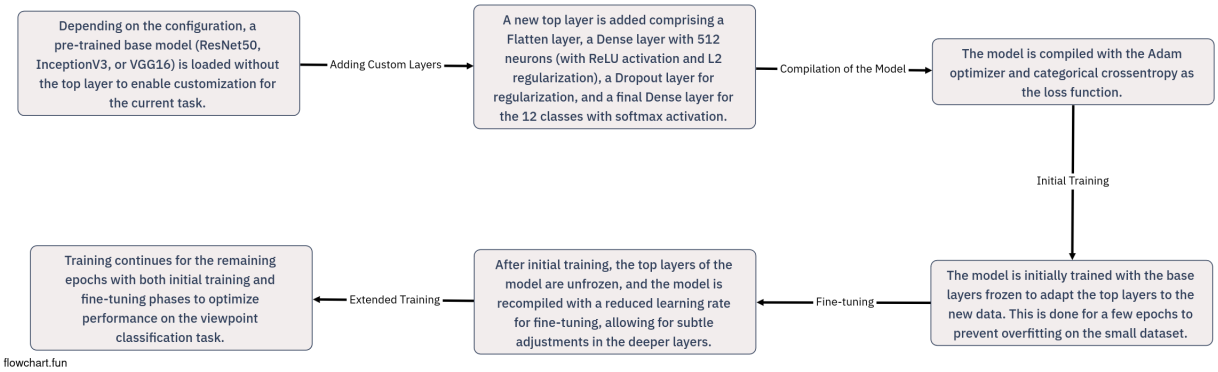

Figure 15: Workflow of Model Training

Table 3: Hyperparameter settings on different models

Model	epochs	learning_rate	data_augmented	img_size	batch_size
ResNet50	20	0.001	False	224×224	32
InceptionV3	20	0.001	False	299×229	32
VGG16	20	0.001	False	224×224	32

6.2 Pre-Trained Model Architecture

6.2.1 ResNet50

ResNet, which stands for Residual Networks, was introduced by He et al. in 2015 He et al. [2016]. It is famous for its "residual blocks" which help in training much deeper networks than was previously feasible. ResNet50 is a variant with 50 layers deep.

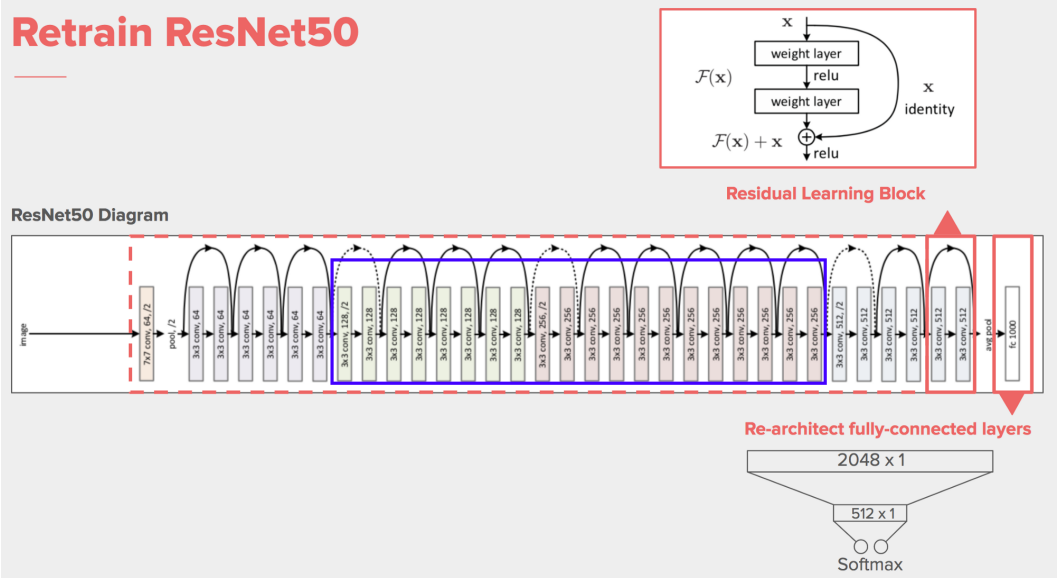

Figure 16: model architecture of ResNet50, from darth manav [2020]

6.2.2 InceptionV3

Developed by Szegedy et al. from Google Szegedy et al. [2015], InceptionV3 is an iteration of the first Inception architecture which was introduced as GoogLeNet during the ILSVRC2014 competition. It is known for its efficiency in terms of computation and memory usage.

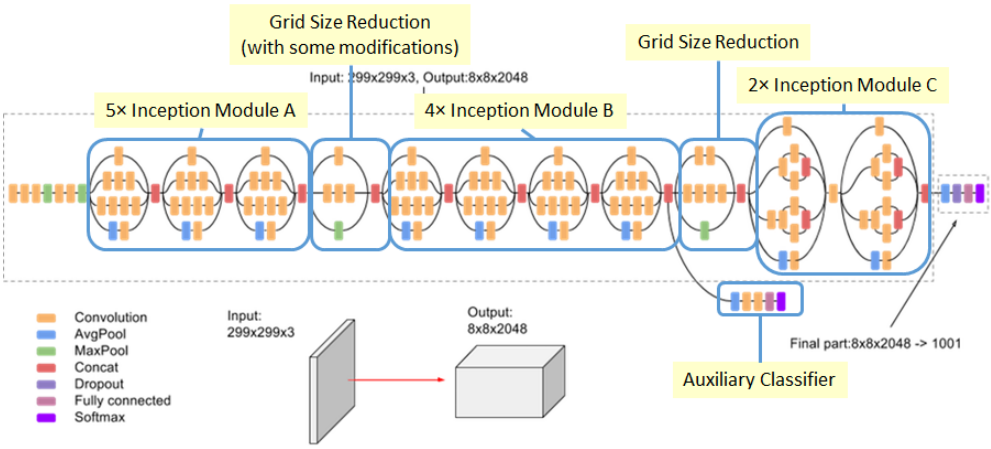

Figure 17: model architecture of InceptionV3, from darth manav [2020]

6.2.3 VGG16

Developed by Simonyan and Zisserman from the University of Oxford Simonyan and Zisserman [2015b], VGG16 was introduced in 2014. It is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth.

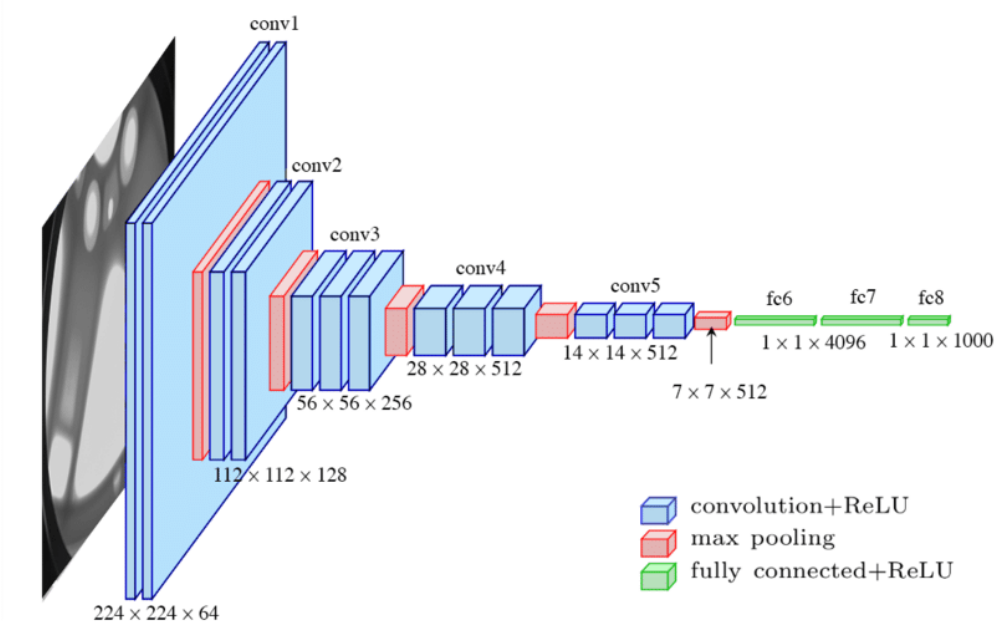

Figure 18: model architecture of VGG16, from darth manav [2020]

6.3 Results

The use of transfer learning for the classification of car viewpoints leverages the pre-existing knowledge embedded in models trained on vast and diverse image datasets. This methodology not only enhances the efficiency of the model training process but also ensures robust performance even with limited data, making it an ideal approach for this project's objectives.

Table 4: Test performance of the tuned model against different models

Model	Validation Accuracy	Training Accuracy
ResNet50	**0.8103**	**1**
InceptionV3	0.5431	0.7725
VGG16	0.6552	0.6695

6.4 Challenges and Solutions

6.4.1 Considerations for Data Augmentation on Viewpoint Classification

In the process of optimizing our machine learning model for the classification of car viewpoints, we encountered challenges with the implementation of data augmentation. Our initial hypothesis was that augmenting our limited dataset would enhance the model's ability to generalize to new, unseen data by presenting a more diverse range of training examples.

6.4.2 Data Augmentation Techniques and Their Implications:

1. Random Flip:
   We considered horizontal flips for augmentation. However, this technique posed a risk to viewpoint classification integrity, as it could potentially reverse the apparent direction of the car. This transformation would only be appropriate if our labels were symmetrical or designed to account for such changes, which was not the case in our project.

2. Random Rotation:
   Minor rotations were applied to simulate variations in the viewing angle, emulating real-world discrepancies that could occur during image capture. Significant rotations were avoided to prevent drastic changes in the perceived viewpoint, which could mislead the model.
3. Random Zoom:
   We implemented random zooms to condition the model to recognize cars at varying scales. This technique was deemed relatively safe and useful for improving scale invariance in the model's predictions.
4. Random Translation:
   Translation helped the model become less sensitive to the precise positioning of cars within the image frame. We imposed limits on the extent of translation to ensure that the cars remained within a reasonable boundary in the images.
5. Random Shear:
   Shearing was initially considered but ultimately excluded from our augmentation strategies. It was determined that shearing could unnaturally alter the vehicle's geometry, creating unrealistic training examples that would not align with typical photographic conditions.

6.4.3 Results of Data Augmentation:

Upon implementing these augmentation strategies, we observed a notable impact on model performance. As shown in Table 3, the ResNet50 model with data augmentation (ResNet50+DA) exhibited a validation accuracy of 0.7414, which was lower than the validation accuracy of 0.8103 achieved by the same model without data augmentation. This was a counterintuitive finding, as it was expected that data augmentation would aid in improving the model's generalization capabilities.

Table 5: Test performance of the tuned model against different models

Model	Validation Accuracy	Training Accuracy
ResNet50	0.8103	1
ResNet50+DA	0.7414	0.9313

6.4.4 Conclusion:

This experiment has been insightful, highlighting that data augmentation is not a one-size-fits-all solution, particularly for tasks requiring precise feature recognition, such as viewpoint classification. We learned that an in-depth understanding of how each augmentation might affect the data is essential, and not all augmentations are beneficial for every type of data or task. Moving forward, we chose to utilize the existing dataset without augmentation, aiming to optimize the model's performance within the given timeframe. This decision aligns with our objective to develop a reliable model tailored to the unique characteristics of our dataset and the specific challenge of classifying car viewpoints.

7 Car Detection and Viewpoint classification (q8)

7.1 Implementation Details:

Car Detection:
We use the YOLOv8s to process each image in the folder and identify the bounding boxes of cars with the class ID 2. The detection includes the position and confidence level for each car.

Viewpoint Estimation:
For each detected car, we crop the car's region from the image and then resize it to fit the input dimensions required by the viewpoint classifier. The classifier, a ResNet50 model, has been trained to predict the angle or viewpoint of the car. It outputs a class that corresponds to a particular angle, the system classifies the orientation of each detected car. The prediction is then converted into an angle.

Visualization:
Matplotlib is used to create a plot where the original image is shown with the overlaid bounding

boxes and arrows indicating the viewpoint orientation. For every detected car, an arrow is plotted to show the estimated viewpoint. The arrow's direction is based on the predicted class from the viewpoint classifier, translating to an angle that represents the car's orientation.

7.2 Results:

Figure 19: combined results of Car Detection and Viewpoint classification model

Appendix

Below are the metrics used to evaluate the performance of the U-Net model for road classification. Blue label is the base line U-Net model, while orange label is U-Net with fused depth mask.

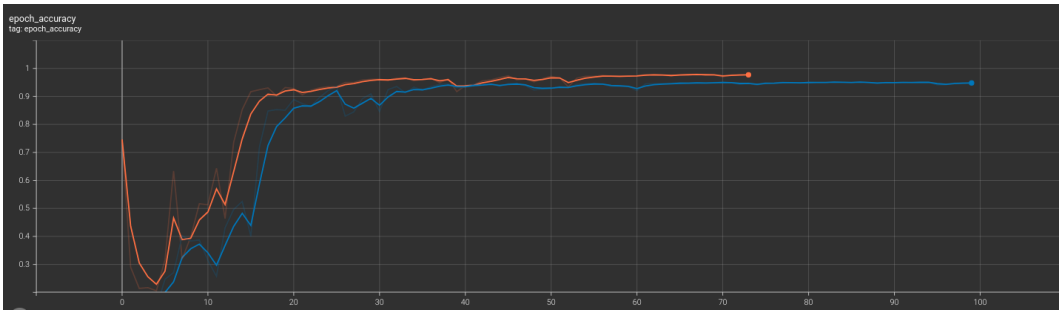

Figure 20: Evaluation accuracy of U-Net model for road classification

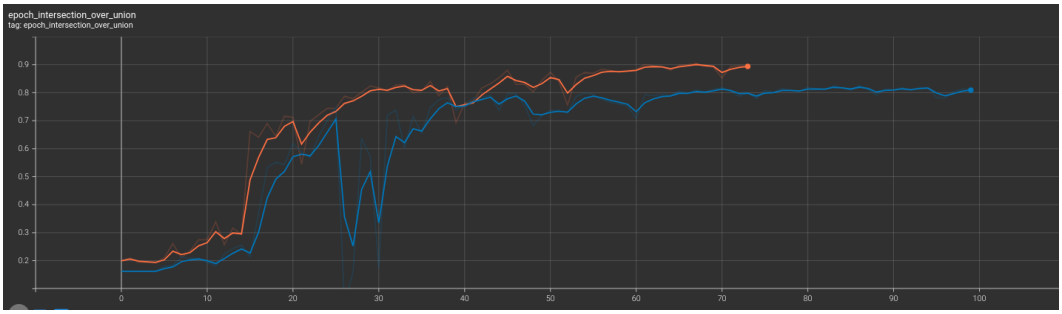

Figure 21: IoU of U-Net model for road classification

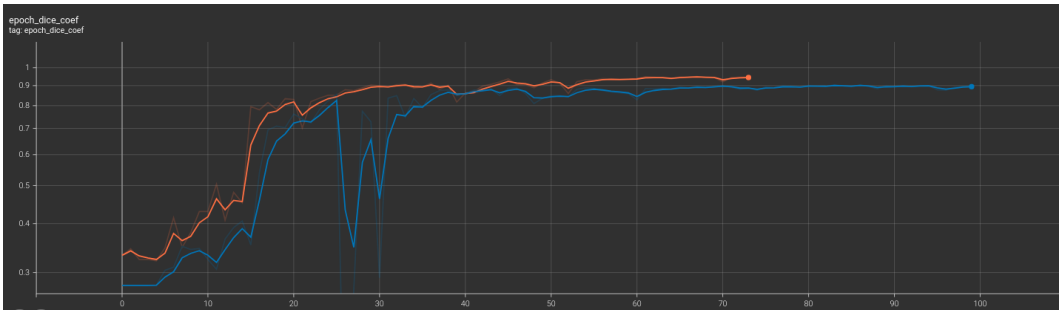

Figure 22: Dice Coefficient of U-Net model for road classification

YOLOv5 and YOLOv8 Car Detection Results on Kitti Dataset:
YOLOv5: `https://youtu.be/TIHbHYUT2AU`
YOLOv8: `https://youtu.be/QrtF-zzVeI0`

References

Qiang Liu, Ruihao Li, Huosheng Hu, and Dongbing Gu. Extracting semantic information from visual data: A survey. *Robotics*, 5(1), 2016. ISSN 2218-6581. doi: 10.3390/robotics5010008. URL https://www.mdpi.com/2218-6581/5/1/8.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Glenn Jocher. YOLOv5 by Ultralytics, May 2020. URL https://github.com/ultralytics/yolov5.

Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, January 2023. URL https://github.com/ultralytics/ultralytics.

Jan 2023. URL https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8.

Xiongwei Wu, Doyen Sahoo, and Steven C. H. Hoi. Recent advances in deep learning for object detection, 2019.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

Feb 2023. URL https://www.augmentedstartups.com/blog/yolov8-vs-yolov5-choosing-the-best-object-detection-model.

Haiying Liu, Fengqian Sun, Jason Gu, and Lixia Deng. Sf-yolov5: A lightweight small object detection algorithm based on improved feature fusion mode. *Sensors*, 22(15), 2022. ISSN 1424-8220. doi: 10.3390/s22155817. URL https://www.mdpi.com/1424-8220/22/15/5817.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi: 10.1109/TKDE.2009.191.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

darth manav. Resnet 50 , vgg 16, inception v3 -beginner's guide, 2020. URL https://www.kaggle.com/code/darthmanav/resnet-50-vgg-16-inception-v3-beginner-s-guide. [Online; accessed April 27, 2013].

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015b.