# EN.601.661: Computer Vision Project Midpoint Check-In

Title: Multi-Task Perception Model for Autonomous Vehicles

Simultaneous Object Detection, Semantic Segmentation, and Depth Estimation

**Spring 2025**
**April 7, 2025**

## Team Members


- Minh Lê (mle20)
- Yi Lu (ylu174)
- Ritwik Rohan (rrohan2)
- Yuhan Yong (yyong2)


## Introduction

- **Goal:** Develop a unified multi-task perception model that can perform semantic segmentation, depth estimation, and object detection concurrently using a shared encoder-decoder architecture for autonomous driving.
- **Motivation:** Running these perception tasks independently is computationally inefficient and too slow for real-time deployment in autonomous vehicles. Multi-task models reduce redundancy, speed up inference, and improve feature sharing across tasks.
- **Related Work:**
  - **Nekrasov et al. (2019)** [1] proposed a real-time joint model for semantic segmentation and depth estimation using a lightweight encoder and asymmetric annotations. They demonstrated that multitask learning can reduce inference cost and maintain good accuracy. We build on their segmentation-depth backbone and add an object detection branch.
  - **Kendall et al. (2018)** [2] introduced the idea of using task uncertainty to weigh multiple loss functions in multi-task learning. This inspired our approach of loss weighting to balance segmentation, depth, and detection during training.
  - **Mullapudi et al. (2018)** [3] presented *HydraNets*, a dynamic architecture for multi-task learning where a shared encoder branches into specialized decoders. This provides the structural foundation for our model: a shared MobileNetV2 encoder and three task-specific decoders.
- **Our Contribution:** We aim to develop a unified multi-task model capable of performing semantic segmentation, depth estimation, and object detection concurrently without significantly compromising the accuracy of any individual task. To achieve this, we leverage pretrained weights from a strong two-task segmentation-depth model, and extend it by integrating object detection heads (YOLOv8 and SSD). Using knowledge distillation, we ensure that the shared encoder and task-specific decoders are able to learn compact, robust representations across tasks. Our goal is to demonstrate that the integrated model can match the standalone models in accuracy while achieving greater efficiency and speed, thereby validating its suitability for real-time autonomous driving scenarios.

## Methods

### Dataset

For training and evaluating our multi-task perception model, we use the **KITTI Vision Benchmark Suite** [4], which contains high-quality data suitable for semantic segmentation, depth estimation, and 2D object detection. We utilize the following subcomponents:

- **KITTI Semantic Segmentation** – provides pixel-level annotations in Cityscapes format. We remap all labels to a consistent 7-class scheme: `road`, `car`, `vegetation`, `building`, `sky`, `sidewalk`, `fence`.
- **KITTI Depth Prediction** – contains sparse depth maps generated from Velodyne LiDAR, provided as 16-bit PNGs scaled by 256. We retain only valid (non-zero) pixels and match the predicted depth map scale for evaluation.

- **KITTI 2D Object Detection** – provides bounding box annotations for various objects in traffic scenes. For consistency with segmentation classes, we focus on detecting the "car" class in our current experiments.

**Preprocessing:** Minimal preprocessing is needed for segmentation since KITTI follows Cityscapes format. Segmentation ground truth masks are remapped to 7 specific RGB color classes and converted into class index maps. Ambiguous or infrequent classes are marked with the ignore index (255). For depth estimation, only non-zero pixels are used as valid ground truth. For detection, annotations are converted to normalized YOLO/SSD format for training integration.

**Postprocessing:** Model outputs are adapted to KITTI's evaluation formats. Segmentation predictions are mapped back to color images for visualization and class indices for metric evaluation. Depth maps are rescaled and saved as uint16. Zero-mask filtering is applied during metric computation. Predicted bounding boxes (once implemented) will be evaluated with mAP using standard IoU thresholds.

## Methods

- **Implemented Pipeline:** We first preprocessed the KITTI dataset (as described above) and evaluated a pretrained multitask model from Nekrasov et al. [1], which uses a **shared MobileNetV2 encoder** and separate **Lightweight RefineNet decoders** for semantic segmentation and depth estimation. After postprocessing the model outputs to align with KITTI's ground truth (class remapping and depth rescaling), we evaluated segmentation performance using mean Intersection over Union (mIoU) and depth estimation using standard metrics: Absolute Relative Error (AbsRel), Root Mean Squared Error (RMSE), and $\delta$ accuracy thresholds. Separately, we also tested standalone YOLOv8 [6] and SSD [5] models pretrained on the COCO dataset to compute object detection metrics (mean Average Precision, mAP). These single-task models serve as our reference baselines—our goal is to achieve comparable performance across all three tasks using a unified, multitask architecture.
- **YOLOv8 Integration (In Progress):** Our idea is to connect the MobileNetV2 encoder with a YOLOv8 detection head. YOLOv8 requires multi-resolution features for its feature pyramid; we extract encoder outputs at layers corresponding to 80x80, 40x40, and 20x20 resolutions. We pass these features into the YOLOv8 neck and head modules for bounding box regression and classification. This design allows joint learning of all three tasks without compromising speed or memory. The architecture preserves segmentation and depth outputs using the RefineNet decoder and synchronizes detection output through YOLOv8 layers.
- **SSD Integration (In Progress):** Our idea is to integrate the SSD detection head by creating multiscale feature maps from the encoder output. Using MobileNetV2's deep layers (e.g., 19x19, 10x10, 5x5), we build the SSD detection pipeline with additional convolution layers that act as detection heads. Each layer is trained to predict bounding boxes and class probabilities using default anchor boxes. We incorporate L2 normalization and feature adaptation layers to ensure consistency in feature scales. This design enables a lighter-weight detection head compared to YOLOv8.

## Algorithms

- **Knowledge Distillation:** Following Nekrasov et al. [1], we implement a teacher-student learning framework to address incomplete or asymmetric annotations in multi-task learning. Pretrained single-task networks act as teachers, generating pseudo-labels for segmentation and depth estimation tasks to train the student multitask model more effectively.
- **Transfer Learning:** We initialize the task-specific decoders (segmentation, depth, detection) using pretrained weights from respective models. The shared MobileNetV2 encoder is partially frozen during training to preserve its low-level feature representations while allowing deeper task-specific layers to adapt.
- **Loss Balancing:** To address the challenge of optimizing multiple losses jointly, we adopt the uncertainty-based weighting strategy from Kendall et al. [2]. This method dynamically adjusts the contribution of each task's loss during training, enabling better convergence and avoiding overfitting to any single task.

### Software

- **Framework:** We use **PyTorch** (Python 3.10) with CUDA acceleration for deep learning model development. The architecture is fully modularized to support shared encoders and interchangeable task-specific decoders for multitask learning.
- **Evaluation Libraries:** Key libraries include **scikit-learn** (confusion matrix and classification metrics), **NumPy** (numerical operations and metric computation), **OpenCV** and **PIL** (image reading, resizing, format conversions), and **Matplotlib** for qualitative visualizations.
- **Training Infrastructure:** Training and inference are managed through reproducible and customizable pipelines. **YOLOv8** [6] is integrated via the Ultralytics PyTorch Hub, while **SSD** is implemented through the torchvision object detection module as originally proposed in [5]. We utilize PyTorch Lightning-style checkpointing, logging, and visualization to monitor learning curves and loss dynamics.

## Results So Far

- We have implemented and evaluated standalone models for each task to establish baselines. Semantic segmentation and depth estimation were evaluated using the pretrained shared encoder model from Nekrasov et al. [1], which uses MobileNetV2 + RefineNet decoders. Object detection performance was measured using pretrained YOLOv8 weights from Ultralytics [6] and SSD model pretrained on COCO [5]. These were run independently to obtain baseline metrics on our remapped KITTI dataset. Our integrated multitask model will aim to match or approach these results.

| | |
|---|---|
| **Pixel Accuracy** | 0.9643 |
| **Mean Accuracy** | 0.8262 |
| **Mean IoU** | 0.7901 |
| **Frequency Weighted IoU** | 0.9311 |

Table 1: Segmentation Evaluation Metrics

| | |
|---|---|
| **Abs Rel** | 0.1043 |
| **Sq Rel** | 0.4518 |
| **RMSE** | 3.4242 |
| **MAE** | 1.8252 |

Table 2: Depth Evaluation Metrics

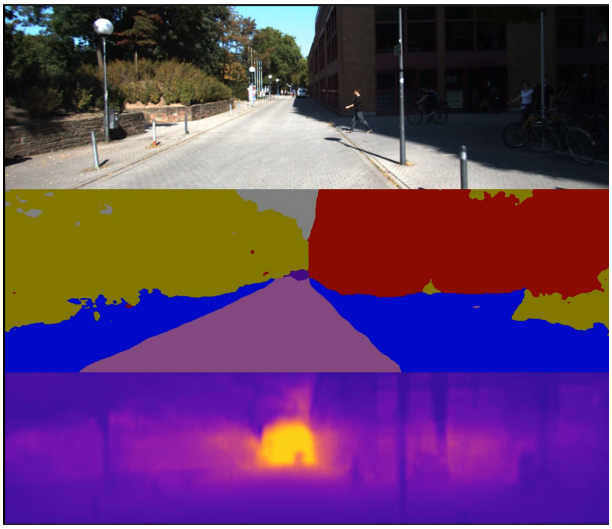

Figure 1: Segmentation and Depth Evaluation Results

| Class | Precision | Recall | AP |
|---|---|---|---|
| Car | 0.563 | 0.854 | 0.694 |
| Truck | 0.267 | 0.571 | 0.390 |
| Van | 0.200 | 0.061 | 0.110 |
| Pedestrian | 0.427 | 0.761 | 0.570 |
| Mean AP: 0.441 | | | |

Table 3: YOLOv8 Object Detection Metrics (IoU threshold = 0.50)

| | |
|---|---|
| **Average RMSE Error** | 22.47 pixels |
| **Median RMSE Error** | 16.76 |
| **Average Normalized RMSE Error** | 0.1489 |
| **Median Normalized RMSE Error** | 0.1113 |

Table 4: SSD Object Detection Evaluation Metrics


Figure 2: SSD Object Detection Results


Figure 3: YOLO Object Detection Results detecting car at a far distance

- After completing standalone evaluation, we are currently working on training the integrated multi-task model with shared encoder and three heads. One version integrates YOLOv8 with segmentation and depth, and the other integrates SSD with segmentation and depth. These results will be compared against the baseline metrics for all three tasks (Table 5). Figure 4 is currently a placeholder for our output that will include RGB input, segmentation, depth, and detection stacked together.

| **Metric** | **Standalone Seg+Depth** | **Standalone YOLO** | **Standalone SSD** | **Multi-task YOLO** | **Multi-task SSD** |
|---|---|---|---|---|---|
| **Mean IoU** | 0.7901 | Not applicable | Not applicable | In Progress | In Progress |
| **RMSE** | 3.4242 | Not applicable | Not applicable | In Progress | In Progress |
| **Abs Rel** | 0.1043 | Not applicable | Not applicable | In Progress | In Progress |
| **mAP@50 (YOLO)** | Not applicable | 0.441 | Not applicable | In Progress | Not applicable |
| **mAP@50 (SSD)** | Not applicable | Not applicable | 0.1489 | Not applicable | In Progress |

Table 5: Comparison of standalone and multi-task models across segmentation, depth estimation, and object detection tasks (YOLOv8 and SSD).
*Note: This is a demonstration table. Our final report will include a more comprehensive and suitable version based on the complete experimental results.*

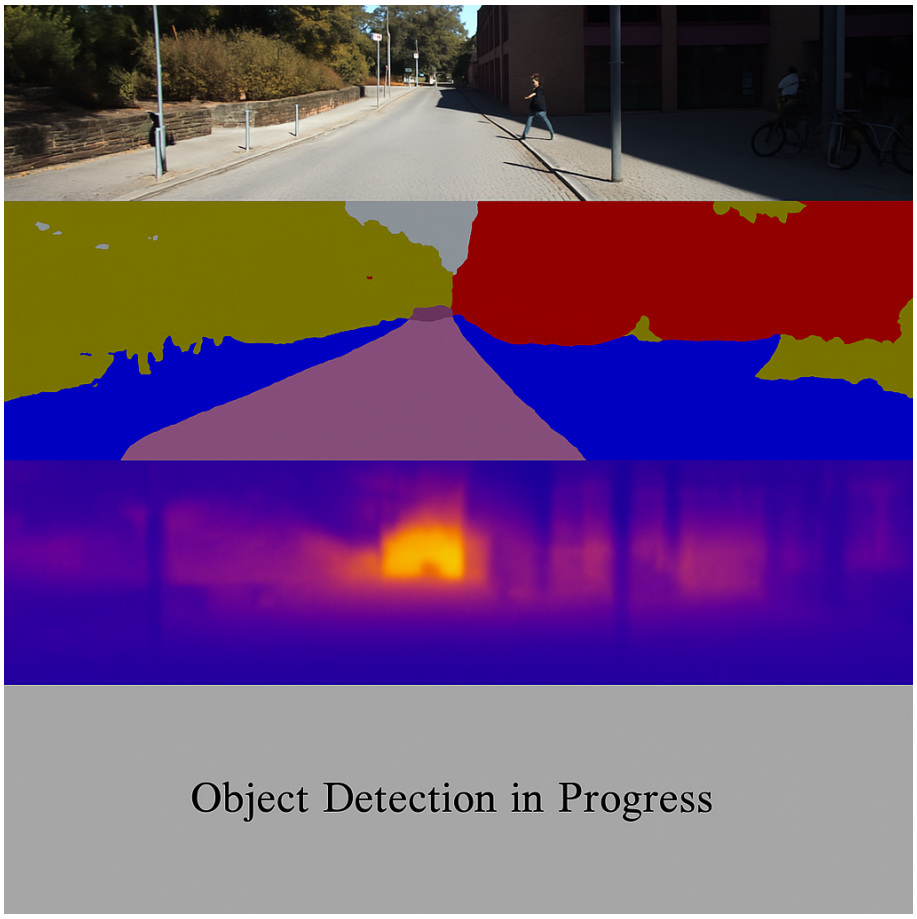


Figure 4: Demo output which will include simultaneous object detection output as well.

# Discussion

## 1. Summary of Results

- Our standalone models have shown promising results on the KITTI dataset, providing a solid baseline for multitask integration. The pretrained segmentation + depth model from Nekrasov et al. [1] achieves a mean IoU of 0.79 and RMSE of 3.42 with AbsRel 0.104 on depth.
- For object detection, the standalone YOLOv8 model (Ultralytics [6]) shows strong performance in detecting cars with high accuracy and fast inference, while the SSD model [5] provides reasonable accuracy with significantly lighter computation.
- These results validate the use of KITTI for multitask benchmarking and give us clear quantitative targets.
- We are currently in the process of training our integrated multitask model that combines semantic segmentation, depth estimation, and object detection (using either YOLOv8 or SSD heads) into a single shared encoder-decoder framework. Once complete, we will compare the multitask model's performance with each of the above standalone baselines to verify performance retention.
- Additionally, we aim to compare which detection head integrates more efficiently into the multitask pipeline in terms of both accuracy and inference speed.

## 2. Comparison to Other Work

- Our segmentation and depth results are comparable to those in Nekrasov et al. [1], despite differences in class mappings and dataset splits. Unlike Nekrasov's model, we extend the architecture to include a third task object detection using either YOLOv8 or SSD.
- Previous multitask models (such as HydraNets [3]) typically focused on two tasks or relied on highly customized decoders. Our approach focuses on modular, extensible architecture that supports multiple detection heads and can scale to 3 or more tasks.
- By maintaining performance close to that of standalone models, our framework demonstrates that multitask learning can achieve both efficiency and accuracy without major compromises a critical requirement in autonomous driving applications.

## 3. Planned Improvements

- We are still training the integrated multitask models, as detailed in the Methods section. This includes adapting feature fusion strategies between the encoder and decoders.
- Depending on training progress and evaluation, we may need to refine postprocessing or adjust the weighting strategy in the multitask loss to ensure stability.
- If time permits, we also plan to experiment with integrating a third object detection head based on PointPillars [7], which uses LiDAR data. This would allow us to compare monocular-based detection heads (YOLO/SSD) with LiDAR-based methods.
- Additional improvements may include profiling model FPS, GPU memory usage, and real-time feasibility especially important for embedded deployment in autonomous vehicles.

# Conclusion (Projected)

- Our standalone evaluation confirms that pretrained segmentation, depth, and detection models provide strong performance on the KITTI dataset, offering reliable baselines.
- The shared encoder-decoder architecture has proven effective for combining semantic segmentation and depth estimation, showing high accuracy with efficient inference.
- Our integrated model aims to extend this success to include object detection, creating a unified perception model for autonomous vehicles.
- We intend to match or improve the individual performance metrics of the standalone models when all three tasks are trained jointly. Achieving this will demonstrate that our multitask architecture does not compromise on accuracy while significantly reducing redundancy and inference cost.
- Once training is complete, we will compare both the accuracy and speed of the integrated model against the standalone pipelines to assess real-world feasibility for multitask perception.

# References

[1] V. Nekrasov, T. Dharmasiri, A. Spek, T. Drummond, C. Shen, and I. Reid, *Real-Time Joint Semantic Segmentation and Depth Estimation Using Asymmetric Annotations*. arXiv preprint arXiv:1809.04766, 2019.

[2] A. Kendall, Y. Gal, and R. Cipolla, *Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics*. arXiv preprint arXiv:1705.07115, 2018.

[3] R. T. Mullapudi, I. Radosavovic, and D. Ramanan, *HydraNets: Specialized Dynamic Architectures for Multi-Task Learning*. CVPR 2018. Available at: `https://openaccess.thecvf.com/content_cvpr_2018/papers/Mullapudi_HydraNets_Specialized_Dynamic_CVPR_2018_paper.pdf`

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, *Vision meets Robotics: The KITTI Dataset*, International Journal of Robotics Research (IJRR), 2013. Available at: `https://www.cvlibs.net/datasets/kitti/`

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*, arXiv preprint arXiv:1512.02325, 2015. Available at: `https://arxiv.org/pdf/1512.02325`

[6] G. Jocher et al., *Ultralytics YOLOv8*, GitHub repository, 2023. Available at: `https://github.com/ultralytics/ultralytics`

[7] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, *PointPillars: Fast Encoders for Object Detection from Point Clouds*. arXiv preprint arXiv:1812.05784, 2018. Available at: `https://arxiv.org/pdf/1812.05784`