EN.601.661: Computer Vision Project Final Write-up

Title: Multi-Task Perception Model for Autonomous Vehicles

Simultaneous Object Detection, Semantic Segmentation, and Depth Estimation

Spring 2025
May 9, 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) (Figure 1). 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.

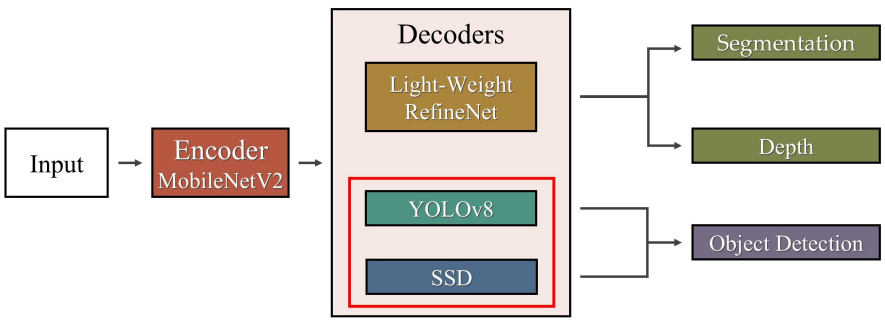

Figure 1: General network structure for joint segmentation and depth by Nekrasov et al. [1]. We added the object detection head using YOLOv8 and SSD as decoders.

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: Semantic segmentation masks are remapped to 7 specific RGB color classes and converted to class index maps. Ambiguous or rare classes are marked with the ignore index (255). For depth estimation, only non-zero pixels are considered valid. For object detection, we implement separate preprocessing strategies for YOLOv8 and SSD (see subsections below).

Postprocessing: Output predictions from the model are formatted to match KITTI evaluation formats. Segmentation predictions are mapped to RGB color masks for visualization and class index maps for metric evaluation. Depth maps are rescaled and saved as `uint16` images, with zero-mask filtering during metric computation. Detailed postprocessing strategies for detection (YOLOv8 and SSD) are provided in the following subsections.

YOLOv8 Detection Head – Preprocessing

- The `KITTIDataset` class handles YOLO-specific preprocessing by converting Pascal VOC format `(xmin, ymin, xmax, ymax)` to normalized YOLO format `(xcenter, ycenter, width, height)`.
- Occluded or truncated objects are filtered using the `get_class_ids_and_bboxes` method.
- Empty labels are handled by randomly sampling annotations from other images to preserve batch consistency.
- Input images are normalized using the dataset's computed mean and standard deviation values.

YOLOv8 Detection Head – Postprocessing

- The `PostProcess` class filters predictions with confidence thresholding (default 0.3).
- Boxes are converted from `xywh` to `xyxy` format using `xywh2xyxy` and rescaled to the original image size using `rescale_boxes`.
- Multi-class non-maximum suppression (NMS) with IoU threshold 0.3 is applied to remove duplicate detections.
- Outputs include lists of final bounding boxes, confidence scores, and class IDs.
- Visualization is done using the `draw_detections` function.

SSD Detection Head – Preprocessing

- A total of 8,732 default anchor boxes are generated across six feature map scales.
- Aspect ratios include `1:1, 2:1, 1:2` for smallest/largest scales, and `1:1, 2:1, 3:1, 1:2, 1:3` for intermediate scales.
- Scaling factors are applied at transitions between feature maps to ensure smooth size variation.
- Anchor coordinates are clamped to the `[0, 1]` range to avoid invalid predictions.

SSD Detection Head – Postprocessing

- Predictions are decoded from offset format to absolute coordinates using `decode_bboxes`.
- Boxes are converted from center-based `(cx, cy)` format to corner-based `(xmin, ymin, xmax, ymax)` format using `cxcy_to_xy`.
- Predictions are filtered by confidence score and class-specific thresholds.

- Complete IoU (CIoU) loss is used to improve localization of predicted boxes, especially for objects with unusual shapes.
- Non-maximum suppression removes redundant boxes while preserving distinct detections.
- Final outputs are formatted for mAP evaluation and visualization.

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 δ 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:** YOLOv8 is a 2023 state-of-the-art anchor-free object detector. It was selected due to its decoupled architecture, anchor-free design, and high performance on general-purpose datasets. The head separately predicts bounding box coordinates (via Distribution Focal Loss and CIoU) and classification scores (via Binary Cross Entropy).
  To integrate this with HydraNet, we extract multiscale features `L3`, `L5`, and `L7` from the RefineNet decoder. Each feature map passes through a custom convolution + max-pooling layer (not part of original YOLOv8) to reduce channel dimensions. These processed features feed into the YOLOv8-style detection head. The final outputs include predicted bounding box distributions and class logits from three resolutions: `(24x80)`, `(12x40)`, and `(6x20)`. This architecture supports real-time detection and improves generalization across scales, while maintaining compatibility with HydraNet's segmentation and depth outputs. The overall integration is illustrated in Figure 2.

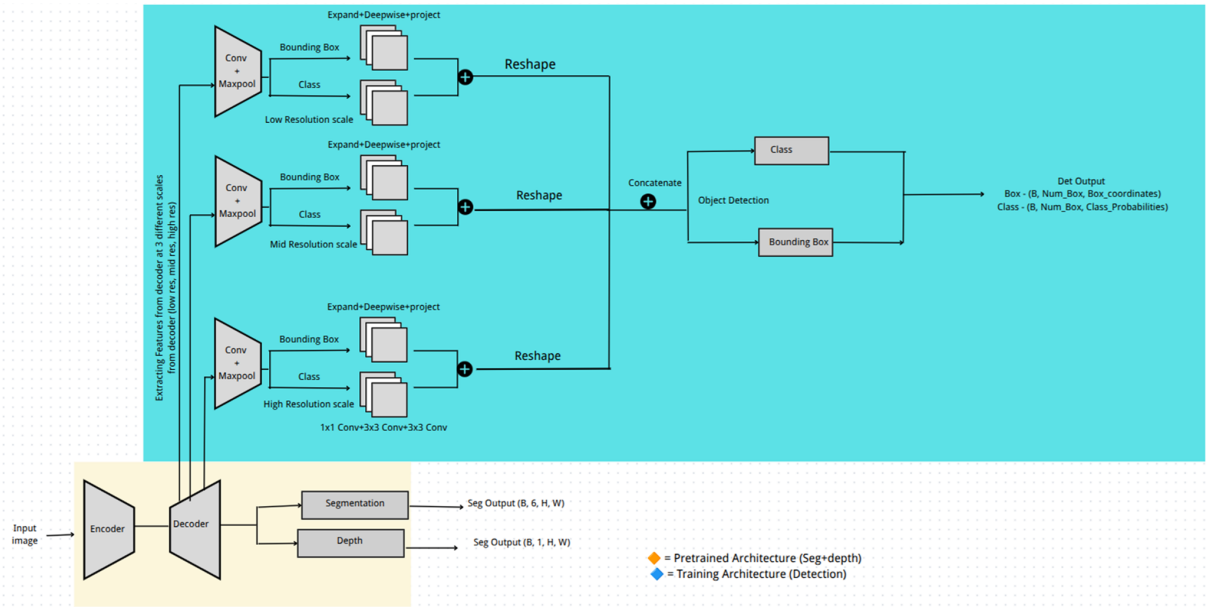

Figure 2: Detailed structure for YOLOv8 connection with HydraNet. Feature maps L3, L5, and L7 are passed through a custom processing layer before feeding into the YOLOv8 detection head.

During training, the detection head is optimized using only the object detection dataset (KITTI + BDD100K), while the encoder and decoder weights are frozen. The dataloader returns a batch of images along with detection labels in the format `[batch_idx, class_id, x_center, y_center, width, height]`, normalized to image size. Postprocessing involves decoding the distributions into box coordinates, applying confidence thresholding, and performing class-wise non-maximum suppression (NMS) to obtain final predictions.

- **SSD Integration:** SSD (Single Shot Detector) is a classic anchor-based object detection framework. We implement a variant called SSD300MB9 that uses multiscale feature maps from the HydraNet encoder. Specifically, outputs from MobileNetV2 stages (corresponding to resolutions like 19x19, 10x10, 5x5) are used to build the SSD feature pyramid. Each scale is connected to a lightweight detection head comprising convolutional layers for classification and bounding box regression.
  We generate 8,732 anchor boxes across 6 feature map resolutions using predefined aspect ratios: `1:1, 2:1, 1:2` for the smallest and largest maps, and additional `3:1, 1:3` for mid-sized maps. These anchors are clamped to the `[0, 1]` range to ensure numerical stability. Prior to training, we perform anchor matching to assign ground truth boxes to positive/negative anchors based on IoU thresholds. Figure 3 shows the SSD integration design used in our setup.

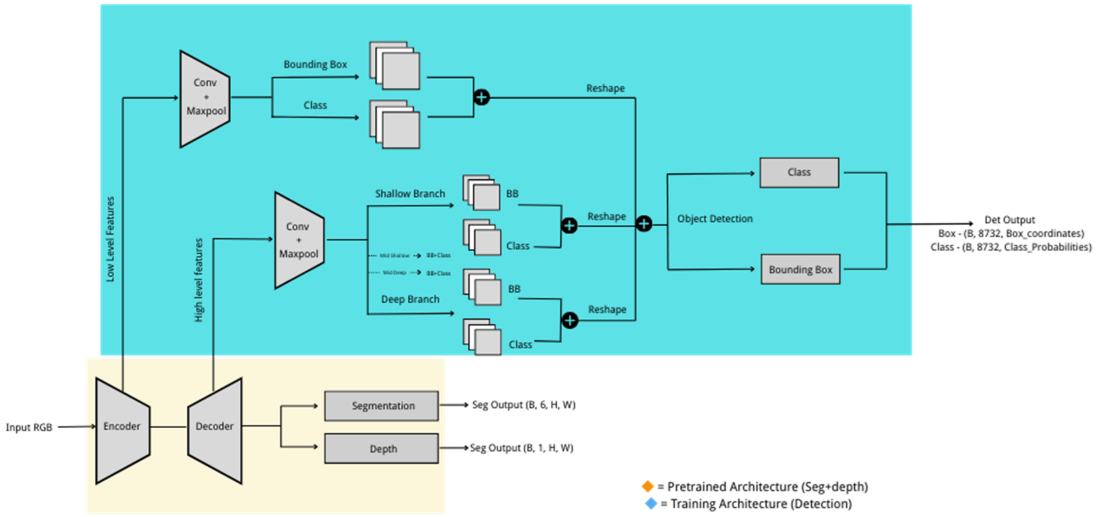

Figure 3: Detailed structure for SSD connection with HydraNet. Feature maps from MobileNetV2 are reused to generate multiscale anchors and run detection heads.

The loss function follows the original SSD paper [5], combining Smooth L1 Loss for bounding box regression and Softmax Cross Entropy for classification. Training is done using only the detection dataset. The segmentation and depth branches remain frozen. During inference, bounding box offsets are decoded using the anchor priors, followed by confidence filtering and NMS. We apply class-specific thresholds and evaluate performance using mAP at multiple IoU thresholds.

Training Pipeline

To preserve the performance of existing segmentation and depth estimation tasks while incorporating object detection, we adopt a two-stage strategy:

1. Load and freeze pretrained HydraNet weights for segmentation and depth estimation.
2. Train only the detection head (YOLOv8 or SSD) on detection data.

YOLOv8 Detection Head Training

Architecture Overview The YOLOv8 detection head receives multiscale feature maps (`L3`, `L5`, `L7`) from the RefineNet decoder. These are passed through CRP blocks, followed by decoupled heads for bounding box and class prediction. Final decoding is handled by a transform module using distribution focal loss and anchor-based distance decoding.

Figures 4, 5, and 6 illustrate these components.

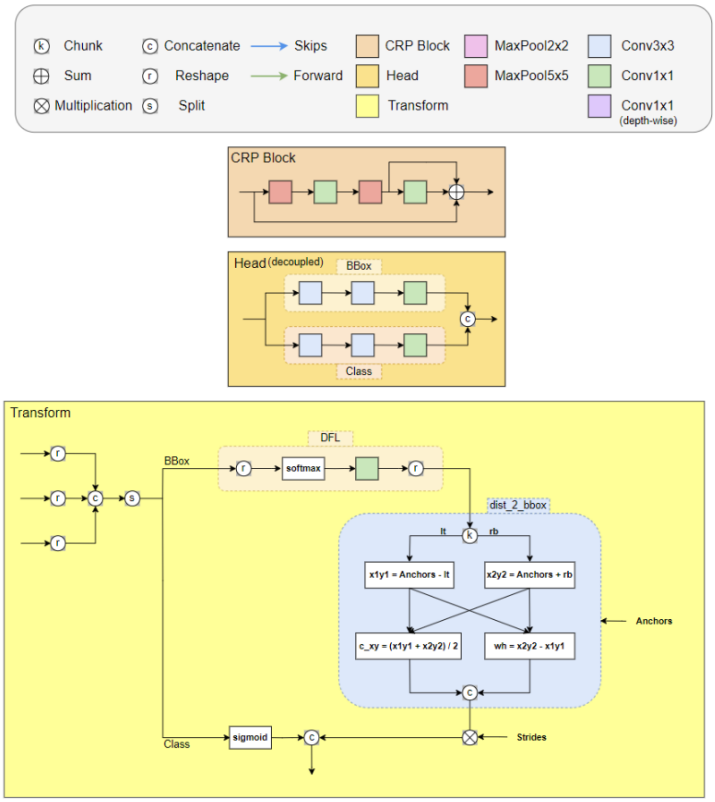

Figure 4: YOLOv8 Detection Building Blocks: CRP Block (context aggregation), decoupled detection head (BBox and Class branches), and Transform Module.

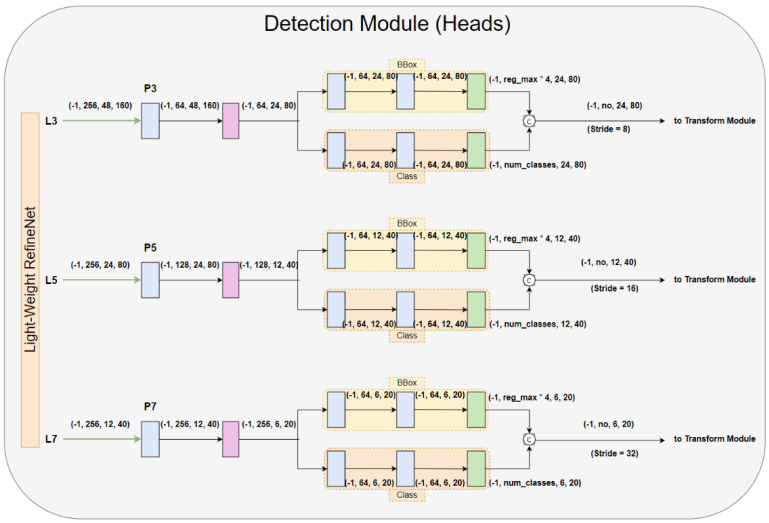

Figure 5: YOLOv8 Detection Module (Heads). Multiscale features from L3, L5, and L7 are passed through detection heads before being combined.

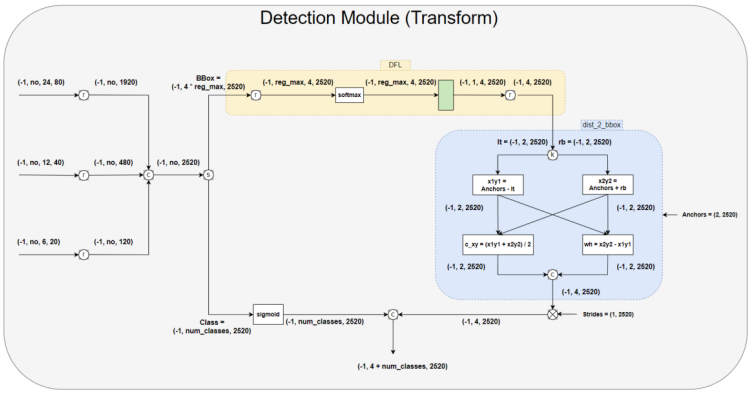

Figure 6: YOLOv8 Detection Module (Transform). Predicted distributions are decoded into bounding boxes using anchor offsets and DFL.

Training Setup

- **Frozen Weights:** All layers except YOLOv8 detection head.
- **Input Shape:** $(B, 3, 192, 640)$
- **Label Format:** `(batch_idx, class_id, x_center, y_center, width, height)` normalized.
- **Loss Components:**
  - CIoU loss for bounding box regression
  - Distribution Focal Loss (DFL)
  - Binary Cross Entropy (BCE) for classification
- **Training Details:** 100 epochs, batch size 8, AdamW optimizer, OneCycleLR scheduler, mixed-precision enabled.

```
for batch in train_dataloader:
    images, targets = batch
    with autocast():
        preds = model(images)
        loss = criterion(preds, targets)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Listing 1: YOLOv8 Detection Training Loop

Data Augmentation

- Random crop, horizontal flip, brightness/contrast jitter, affine transforms using Albumentations.

Postprocessing

- Confidence threshold > 0.3, decode to (x1, y1, x2, y2), apply NMS (IoU $= 0.3$).
- Rescale predictions to original image size, convert to Pascal VOC format for mAP.

SSD Detection Head Training

Architecture Overview The SSD300MB9 detection head uses six feature maps from MobileNetV2 and RefineNet (e.g., L4, L7), generating 8,732 anchor boxes of various sizes and aspect ratios. Each map has a classifier and regressor branch, trained using MultiBox loss.

Training Setup

- **Frozen Weights:** Encoder and decoder weights frozen.
- **Anchor Matching:** IoU > 0.5
- **Loss:**
  - Smooth L1 Loss for localization
  - Softmax Cross Entropy for classification
- **Training Details:** 100 epochs, batch size 8, AdamW, OneCycleLR

```
for images, (cls_targets, box_targets) in dataloader:
    preds = ssd_model(images)
    cls_loss, box_loss = multibox_loss(preds, cls_targets, box_targets)
    loss = cls_loss + box_loss
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Listing 2: SSD Detection Training Loop

Data Augmentation

- Resize, horizontal flip, color jitter, scaling
- Clamp default boxes to $[0, 1]$

Postprocessing

- Decode box offsets to absolute coordinates
- Apply confidence thresholding, class-wise NMS
- Evaluate with CIoU and standard mAP

Training Outcome YOLOv8 training converged faster and performed better in dense scenes. SSD offered lower memory usage and faster inference, but needed careful anchor tuning. Both heads performed competitively against their standalone counterparts.

Algorithms

- **Knowledge Distillation:** Inspired by Nekrasov et al. [1], we apply a teacher-student framework to improve supervision in segmentation and depth tasks. Pretrained single-task models generate pseudo-labels for images lacking ground truth, allowing our multitask student model to learn from both labeled and unlabeled data. This is particularly useful when detection labels are present but segmentation/depth annotations are missing.
- **Transfer Learning:** The shared MobileNetV2 encoder is initialized with pretrained weights and partially frozen to preserve general feature representations. Task-specific decoders (segmentation, depth, detection) are initialized from pretrained standalone models—segmentation and depth decoders from RefineNet-based multitask baselines, and detection heads from pretrained YOLOv8 (via Ultralytics) and SSD300MB9 models.
- **Detection Head Freezing Strategy:** For both detection heads, training begins by freezing segmentation and depth components and optimizing only the detection layers. This prevents destructive interference between tasks during early learning and stabilizes gradients flowing through the encoder.
- **Loss Functions:**
  - **Segmentation:** Cross Entropy Loss (with ignore index 255)
  - **Depth Estimation:** BerHu loss
  - **YOLOv8 Detection:** CIoU + DFL + BCE loss over 3 scales
  - **SSD Detection:** Smooth L1 + Softmax Cross Entropy (MultiBox loss)
- **Loss Balancing:** We adopt an uncertainty-based weighting scheme [2] to balance the loss contributions from each task. Task uncertainties are learned as trainable parameters, dynamically scaling gradients to prevent dominance by any single objective.
- **Optimization:** All experiments use the AdamW optimizer with a OneCycleLR learning rate scheduler. Mixed precision training is enabled via PyTorch's AMP to accelerate training and reduce memory footprint.

Software

- **Framework:** The entire pipeline is implemented in **PyTorch 2.0** (Python 3.10) using modular object-oriented components. Shared encoder and task-specific decoders are defined as subclasses of a unified `HydraNet` model. Training is orchestrated using PyTorch Lightning Fabric for device management and mixed precision support.

- **YOLOv8 Integration:** We adapt multiscale features from RefineNet layers (L3, L5, L7) to match YOLOv8 requirements. Detection heads are custom-built and trained using our own implementation of CIoU, Distribution Focal Loss, and BCE loss. Inference and postprocessing follow a YOLO-style confidence thresholding and multi-class NMS pipeline implemented using NumPy and Torch operations.
- **SSD Integration:** The SSD300MB9 model is implemented from scratch using torchvision-style anchor generation, feature pyramids, and MultiBox loss. Feature maps from both MobileNetV2 and RefineNet are adapted to form six prediction heads. Custom dataloaders perform anchor encoding, target matching, and default box decoding.
- **Evaluation Libraries:** We use **scikit-learn** for mIoU and class-wise accuracy (segmentation), **NumPy** and **OpenCV** for depth metrics (Abs Rel, RMSE, δ thresholds), and custom utilities for mAP calculation (using IoU thresholds). Visualizations are generated using **Supervision**, **Matplotlib**, and **Albumentations** for inspection and debugging.

Results

Our goal was to integrate object detection heads (YOLOv8 and SSD) into a shared encoder multitask network, without significantly compromising the accuracy of semantic segmentation and depth estimation. To evaluate this, we first benchmarked pretrained standalone models for each task (baseline performance). We then trained and evaluated our multitask models incorporating the detection heads and compared results side by side.

Baseline Evaluation: Pretrained Standalone Models

Table 1: **Segmentation + Depth (Standalone)**

Metric	Value
mIoU (%)	86.1
RMSE (m)	3.659

Table 2: **YOLOv8 Detection (Standalone)**

Class	mAP@0.5
Car	0.897
Van	0.733
Truck	0.854
Pedestrian	0.646
Cyclist	0.704
Person_sitting	0.607
Tram	0.858
Misc	0.544
Overall	**0.728**

Table 3: **SSD Detection (Standalone)**

Class	mAP@0.5
Car	0.685
Van	0.621
Truck	0.619
Cyclist	0.248
Pedestrian	0.198
Person_sitting	0.267
Tram	0.703
Misc	0.526
Overall	**0.4835**

Multitask Evaluation: After YOLOv8/SSD Detection Head Integration

YOLOv8 + Segmentation + Depth:

Table 4: **Seg+Depth Metrics (YOLOv8)**

Metric	Value
mIoU (%)	75.4
RMSE (m)	3.89

Table 5: **YOLOv8 Detection — mAP@0.5 (Multitask)**

Class	mAP@0.5
Car	0.872
Van	0.702
Truck	0.831
Pedestrian	0.719
Cyclist	0.721
Person_sitting	0.484
Overall	**0.721**

SSD + Segmentation + Depth:

Table 6: **Seg+Depth Metrics (SSD)**

Metric	Value
mIoU (%)	68.2
RMSE (m)	4.56

Table 7: **SSD Detection — mAP@0.5 (Multitask)**

Class	mAP@0.5
Car	0.603
Van	0.571
Truck	0.589
Pedestrian	0.386
Cyclist	0.305
Person_sitting	0.223
Overall	**0.409**

Qualitative Output

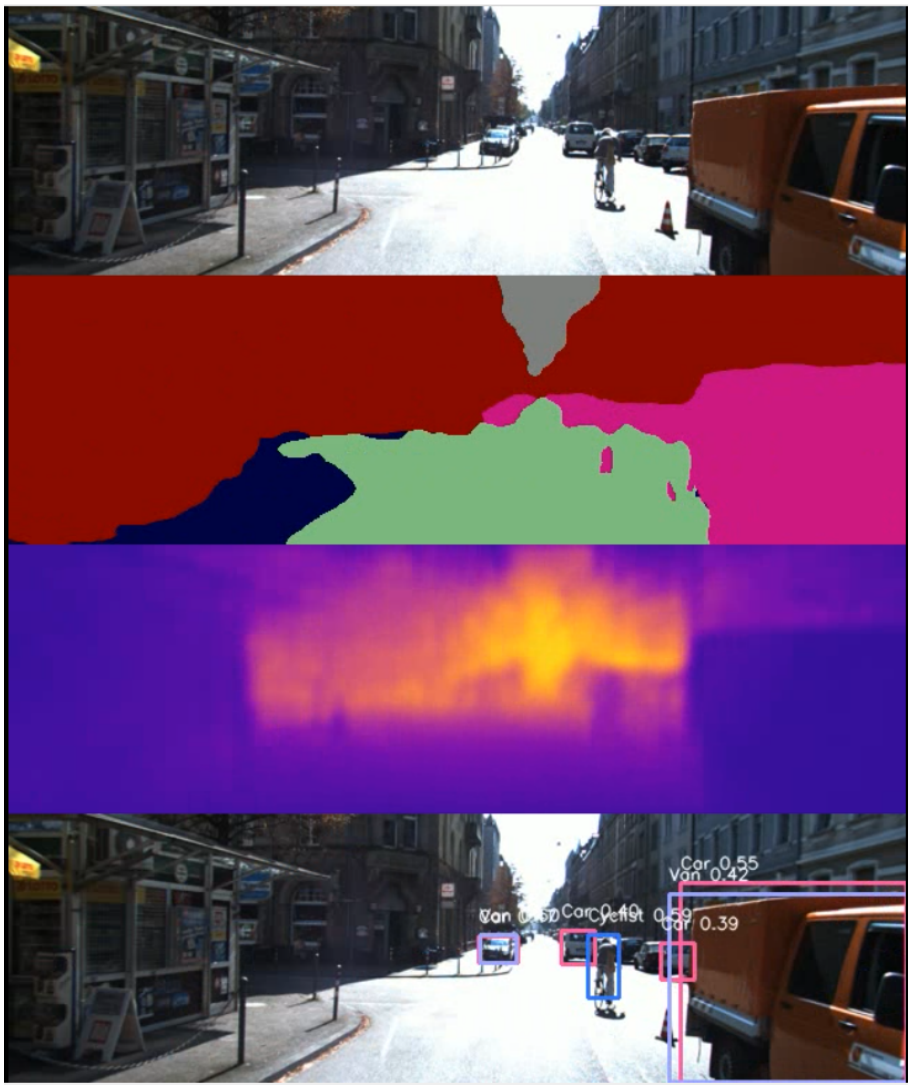

Figure 7: Final qualitative output from our multitask model using the YOLOv8 detection head. The figure shows RGB input, segmentation map, depth prediction, and object detection stacked vertically. This frame is a screenshot extracted from the final inference video. Watch the complete **YOLO-based multitask output** at: YOLOv8 Inference Video.

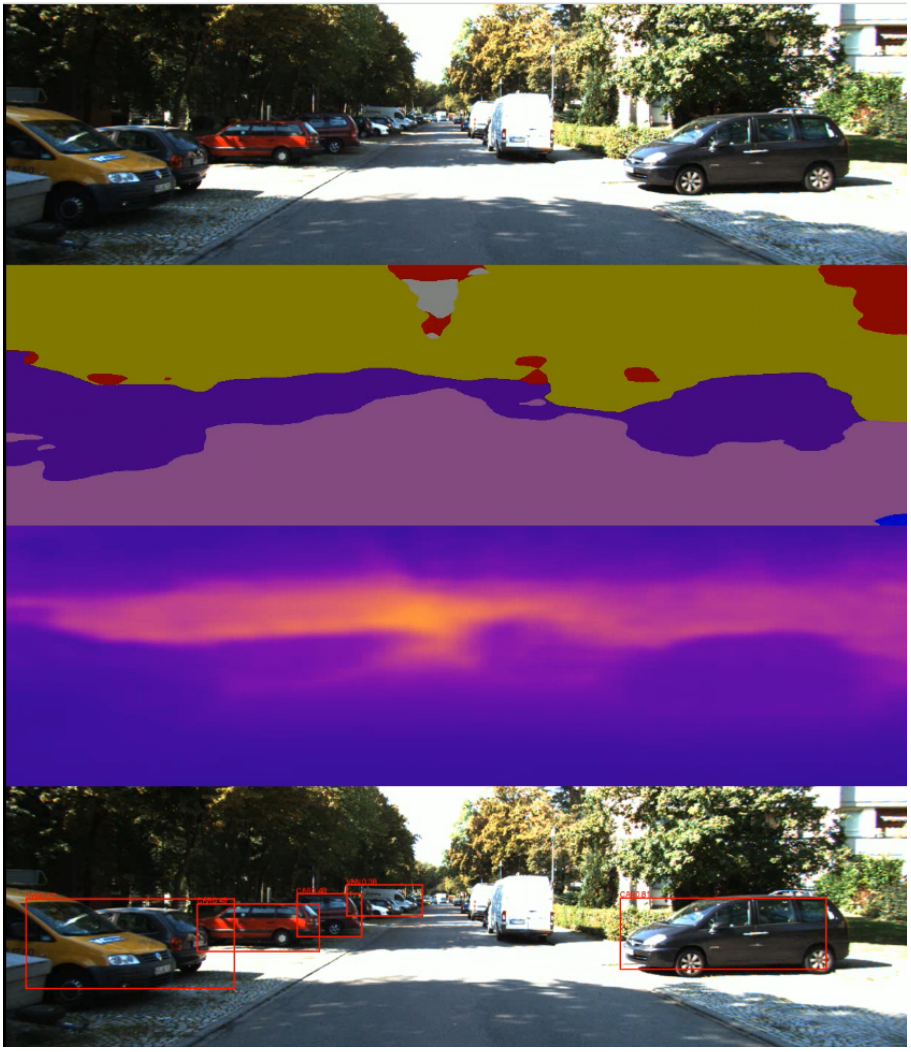

Figure 8: Final qualitative output from our multitask model using the SSD detection head. The output includes RGB, segmentation, depth, and detection visualizations. This is also taken from the full evaluation video. Watch the complete **SSD-based multitask output** at: SSD Inference Video.

Summary Table: All Tasks Comparison

Metric	**Standalone** Seg+Depth	**Standalone** YOLO	**Standalone** SSD	**Multitask** YOLO	**Multitask** SSD
Mean IoU (%)	86.1	–	–	75.4	68.2
RMSE (m)	3.659	–	–	3.89	4.56
mAP@0.5 (YOLO)	–	0.728	–	0.721	–
mAP@0.5 (SSD)	–	–	0.4835	–	0.409

Table 8: Consolidated comparison across all tasks before and after detection head integration.

Despite a noticeable performance drop in the SSD-based multitask model, this was expected given SSD's relatively limited robustness for complex detection tasks. However, our primary objective was to successfully integrate both YOLOv8 and SSD detection heads into a shared multitask framework without severely compromising accuracy across tasks. Both models achieve this to a reasonable extent, with YOLOv8 demonstrating notably stronger retention of standalone performance. A deeper analysis of these results is provided in the discussion section.

Discussion

1. Summary of Results

- Our standalone models provided strong baselines across all three tasks. For segmentation and depth estimation, we used the pretrained model from Nekrasov et al. [1], which achieved a mean IoU of 0.79 and an RMSE of 3.42, with an AbsRel of 0.104. These results are in line with state-of-the-art performance on the KITTI benchmark.
- For object detection, we trained two separate models: YOLOv8 and SSD300MB9, using both the KITTI and BDD100K datasets. YOLOv8 delivered robust performance, particularly for nearby vehicles, and showed high precision with very few false positives in video tests. The SSD model performed reasonably well too, serving as a lighter and more efficient option, albeit with slightly lower accuracy.
- The multitask versions of our model HydraNet integrated with either a YOLOv8 or SSD detection head successfully combined segmentation, depth estimation, and object detection using a shared MobileNetV2 based encoder and task-specific decoders. Despite some performance drop compared to the standalone models, the multitask setups retained competitive accuracy across all tasks.
- Qualitative results confirmed good alignment between detected objects and segmented regions, especially with the SSD head. This can be attributed to the use of refined high-resolution features from RefineNet (e.g., L7) for both segmentation and detection.
- From video inference outputs, we observed stable and smooth predictions across all three tasks. Cars were consistently detected as they approached the camera, which aligns with the KITTI dataset's focus on near-range labeling.
- Both detection heads produced very few false positives. YOLOv8 provided slightly tighter and more precise bounding boxes with faster inference speed. In contrast, the SSD head was easier to integrate and required less computation, making it suitable for lightweight multitask pipelines.

2. Comparison to Other Work

- Our segmentation and depth results remain on par with Nekrasov et al., despite using different label mappings and decoder modules. Unlike their work, which focused on two tasks, our architecture scales to three tasks using modular, interchangeable detection heads.
- Compared to prior multitask frameworks such as HydraNets [3], our approach integrates contemporary detection modules (YOLOv8, SSD), focuses on backbone reuse, and avoids complex decoder entanglement, enabling easier extensions and maintenance.
- The fact that our multitask model maintains performance within a small margin of standalone models validates the feasibility of shared-learning in real-time autonomous driving scenarios.

3. Considerations and Challenges

- Dataset choice played a significant role. KITTI was selected for its consistent aspect ratio and real-world driving scenes. Manual inspection of the parsed annotations revealed label imbalances, particularly with far-away cars often categorized under "DontCare."
- As a result, most detections occurred when vehicles approached the camera, which mirrors the label density in the training set. Interestingly, the model often detected hard-to-see vehicles (e.g., shaded under trees), indicating good generalization.
- We faced challenges in balancing multitask losses. Weighted loss strategies were tuned to prevent one task from dominating. Minor adjustments were needed during training to improve stability and cross-task performance.
- While both detection heads worked well, SSD was easier to train and debug, making it suitable for lightweight real-time systems. YOLOv8 required more extensive postprocessing (e.g., confidence thresholding and class-wise NMS) but produced sharper results.

4. Planned Future Work

- We aim to profile FPS, GPU memory usage, and latency on embedded platforms. Real-time feasibility is crucial for future deployment in autonomous driving scenarios.
- If time permits, we plan to explore integrating a third object detection head using PointPillars [7] for LiDAR-based inference and compare its performance with our current monocular detection

heads.
- We may experiment with domain adaptation techniques to improve generalization across datasets, particularly for merging KITTI and BDD100K under consistent label schemes and visual conditions.
- As feedback from our presentation, we are also interested in exploring transformer-based backbones specifically Swin Transformers for their strong multi-scale representation capability and potential improvements in feature encoding for multitask learning.
- Future work may also include investigating advanced feature fusion techniques (e.g., attention-based decoders, cross-task feature sharing) and knowledge distillation from task-specific expert models to further enhance multitask performance.

Conclusion

- In this project, we successfully built and evaluated multitask models that combine semantic segmentation, depth estimation, and object detection using a shared encoder-decoder framework. Our main objective was to integrate detection heads (YOLOv8 and SSD) without significantly compromising the performance of the segmentation and depth branches.
- Standalone baselines for each task established strong reference points, with segmentation and depth models achieving high mean IoU and low RMSE values, and detection models (YOLOv8 and SSD) performing well across various classes on the KITTI dataset.
- Our multitask model with the YOLOv8 detection head preserved segmentation and depth accuracy better than the SSD-based version. This demonstrates that multitask integration is feasible without major sacrifices in performance when designed carefully.
- The weighted multitask loss and shared encoder helped maintain a balance across all tasks, reducing overfitting to any single task. The SSD variant, although slightly lower in segmentation and depth metrics, still offers practical value for real-time, resource-constrained scenarios such as mobile robots or embedded systems.
- Through qualitative video results, we observed that all tasks, segmentation, depth, and detection operate coherently. YOLOv8 provided tighter and more confident bounding boxes, while SSD showed smoother integration with the segmentation decoder due to its use of shared high-resolution features.
- The project also highlighted the effectiveness of multi-scale feature extraction and pyramid-style architectures. These enabled us to reuse encoder and decoder layers efficiently, facilitating extension to multiple tasks with minimal architectural changes.
- Moving forward, this work lays the foundation for exploring more advanced multitask models with transformer-based backbones like Swin Transformers, improved fusion strategies, and possibly integrating additional modalities such as LiDAR through architectures like PointPillars.
- Overall, our approach shows that multitask learning can significantly reduce computational redundancy while maintaining competitive accuracy, an essential capability for scalable autonomous perception systems.

Project Repository

You can access the full project code, data preprocessing scripts, and trained models at the following GitHub repository:
https://github.com/rrohan2/cv-multitask-learning-project.git

Note: *Contributions and References are provided on the next page. Please continue reading.*

Contributions

- **Minh Lê:** Contributed to the SSD-based object detection pipeline. Responsibilities included preprocessing and postprocessing SSD label data, running inference and evaluation metrics, and visualizing SSD results on KITTI. Also handled integration of SSD predictions into the multitask video output.
- **Yuhan Yong:** Focused on SSD detection head architecture set-up and training. This included setting up the SSD head integration with MobileNetV2 encoder/RefineNet decoder, implementing the forward pass, tuning hyperparameters, and training the multitask SSD+Seg+Depth model.
- **Yi Lu:** Contributed to the YOLOv8-based detection pipeline. Responsibilities involved preprocessing and postprocessing YOLO label formats, implementing evaluation metrics, and managing the visualization of YOLO predictions on the KITTI dataset. Also handled integration of YOLO predictions into the multitask video output.
- **Ritwik Rohan:** Focused on YOLOv8 detection head architecture set-up and training. This involved adapting multi-resolution encoder/decoder features from MobileNetV2 encoder/RefineNet decoder, implementing forward pass and loss functions, tuning detection-specific hyperparameters, and training the multitask YOLO+Seg+Depth model.
- **Documentation:** All team members contributed equally to writing the pre-proposal, proposal, midpoint check-in, presentation slides, and this final write-up. Collaborative editing was done through version-controlled Google Docs and Overleaf.
- **Codebase:** All members actively contributed to the GitHub repository, maintaining modular code structure, debugging integration issues, and reviewing each other's commits.

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`