JASMUR: Estimating the Absolute Pose of Vehicles in Real-World Traffic Images

Jasmine Park
Stanford, NVIDIA
CS231N
jaspark@stanford.edu

Umur Darbaz
Stanford, NVIDIA
CS231N
udarbaz@stanford.edu

Abstract

Vehicle pose estimation is a key component of path planning in self-driving cars [1]. Determining the position of other cars in traffic and predicting their movement by estimating their orientation in 3D space allows a self-driving car to drive safely and avoid collisions. We revisit the Peking University/Baidu Autonomous Driving Kaggle competition [2] focused on estimating the absolute pose of vehicles from RGB real-world traffic images. We present JASMUR, a 6D pose estimator that predicts the location and orientation of cars built on top of state-of-the-art object detection methods. We explore different CNN and transformer feature extractor backbones, define a mean-average-precision metric and achieve up to 0.258 mAP in predicting combined 6D pose with a pretrained ResNet50.

1. Introduction

Vision-based self-driving cars operate on a combination of multiple-cameras (producing RGB images), and a variety of sensors [3]. Input image data provides valuable information such as other vehicles in traffic. Detected vehicles have a translation vector in 3-dimensional space (x, y, z) and a 3-dimensional rotational vector (yaw, pitch, roll). These vectors combined, 6D pose, present the location and orientation of detected objects relative to the camera. Pose is a key component of path planning, as it is not only used to predict behavior of other vehicles, but also pedestrians in the scene [4, 5].

Our objective is to estimate the pose of vehicles (x,y,z, and yaw,pitch,roll) from RGB real-world traffic images. This problem can be split into two parts; (1) an object detection problem, and (2) a pose regression problem. First, every vehicle in a frame must be detected accurately, and in real-time. Next, for each detected vehicle, their pose relative to the camera must be estimated.

2. Related Work

We first evaluate existing top entries for object detection and pose estimation. One of the state-of-the-art vehicle pose estimators is 6D-VNet [1], which uses a Faster-RCNN object detector with a Resnet-101 backbone. The extracted features of the vehicles are run through linear layers to generate a translation vector and a rotation vector. A separate translation loss and rotation loss is calculated from the predictions and ground truth, and the network is trained end-to-end. The paper also shows that the spatial relationship between objects within the image are beneficial to capture using a non-local block at the output of the backbone. 6D-VNet has explores using Euler angles and Quaternions for regressing rotation loss. It additionally considers regressing translation vectors in world coordinates and pixel units. We take their exploration into account when designing our architecture.

Another leading vehicle pose estimation framework, apart from those in the Kaggle competition, is GSNet (Geometric and Scene-aware Network) [6]. GSNet uses an R-CNN object detector and a heatmap regression branch to extract Region of Interest (ROI) features, keypoint coordinates (global locations in the whole image) and bounding boxes. These are fused and fed into three separate heads that regress a translation vector, a rotation vector and reconstruct the vehicle shape. GSNet defines several loss terms that are summed to generate a final loss term, but an important aspect is that Euler angles are restricted to $[-\pi, \pi]$. This ensures each pose can be represented with a unique angle.

CenterNet [7] presents a comprehensive framework for object and keypoint detection. It produces two separate outputs, (1) the center point of detected objects, and (2) a heatmap that contains the probability of a pixel belonging to the center of a detected object. It's a popular choice in the Kaggle competition, and we believe its overall system architecture is applicable to our problem.

YOLO [8, 9, 10] is a state-of-the-art object detection tool that uses a backbone CNN and feature pyramid networks [11] to detect and draw bounding boxes around objects in

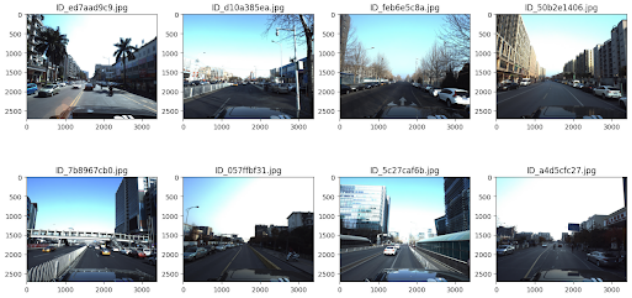

Figure 1. Random samples of the PKU/Baidu dataset visualized.

a given image. Unlike R-CNNs [12], YOLO does not have region proposal. It instead divides the image into a grid and predicts classes of objects and bounding boxes within each cell. Because it works in a single forward pass, YOLO is applicable to real-time problems such as self-driving vehicles. It is an efficient network, containing 36.9M parameters and requiring 104.7 GFlops of operations while achieving 66.7% AP in the case of YOLOv7 [8].

Transformer use in self-driving applications is also increasing. Scene Transformer [13] embeds agents (such as vehicles and pedestrians) into high dimensional spaces and utilize attention to encode interactions between the road and the agents to draw trajectories. Another technique for 6D pose estimation and tracking is presented in FoundationPose [14]. FoundationPose takes in reference images and models of objects (which are encoded as neural representations of appearance and geometry) and uses a transformer network trained with contrastive learning to generally handle many object representations and produce pose estimation.

3. Dataset

For this project, we are utilizing a highly specialized dataset from the Peking University/Baidu Autonomous Driving competition hosted on Kaggle [2, 15]. The dataset includes 60,000 labeled 3D car instances derived from 5,277 high-resolution real-world images as visualized in Figure 1. These images are annotated with 3D orientation labels (yaw, pitch, roll) along with spatial coordinates (x, y, z) for each vehicle in the image. Computer-Aided Design (CAD) models of cars in the images are also provided.

The primary data consists of JPEG format images. These images, derived from real-world scenarios, are crucial for the application of machine learning models that predict the pose and dimensions of cars within the images. Each image is associated with a prediction string that encodes the 3D pose and location of cars within the image. These are the ground-truth labels. An example of the prediction strings is shown in Table 1. These strings are crucial for training and evaluating our pose estimation models. Additionally, mask

ID	Yaw	Pitch	Roll	x	y	z
28	0.169264	0.00461133	-3.1264	-2.52194	3.94	16.6459
23	0.146421	-0.0302788	-3.0741	-3.08984	7.33516	37.4124
43	0.157318	3.12389	-3.10215	-4.93734	9.87454	58.4607

Table 1. Sample labels for each vehicle in an image.

images are provided for each instance to facilitate the focus on relevant regions during processing.

4. Methods

JASMUR employs a combination of object detection, image preprocessing, data augmentation and pose regression techniques to tackle 6D pose estimation. The full pipeline is illustrated in Figure 2. The pipeline takes in 3384x2710 RGB images of real-world traffic data. Every vehicle in each image has an associated 6D pose label (x, y, z, yaw, pitch, roll). Our end-to-end pipeline predicts the 6D pose for each vehicle in an image. Next, we describe the design decisions made for every stage of the pipeline.

4.1. Object Detection

We utilize YOLOv8, a state-of-the-art object detection model to accurately identify and localize cars within each image. YOLOv8 strikes a balance between accuracy and computational performance which makes it a popular choice in real-time applications such as self-driving vehicles. Each input image is fed into YOLOv8, which in turn outputs detected objects' classes and bounding boxes around each object as two vertex coordinates (x_1y_1, x_2y_2) in 2D image space. Using the camera matrix, we map the 3D coordinate labels (x, y, z) of each vehicle onto the 2D image space. Once the coordinates are transformed, we match each image-space coordinate with a corresponding YOLO-detected bounding box. We enumerate the steps taken to associate the bounding boxes with the labels below.

1. The center-point (x_cy_c) of each bounding box is calculated from the YOLO-detected bounding box vertices.
2. The 3D coordinate labels (x, y, z) of each car is mapped to 2D image space (x_ly_l) for every car instance per image.
3. The car coordinates (x_ly_l) are compared to each bounding box center-point (x_cy_c) to find the closest one using Euclidean distance.
4. The bounding box with the smallest distance to the projected 2D center point is considered a match.

This process yields a list of objects with their associated bounding box and pose labels. The bounding box is used to crop the image around the detected car, ensuring that only the region of interest is used for further processing.

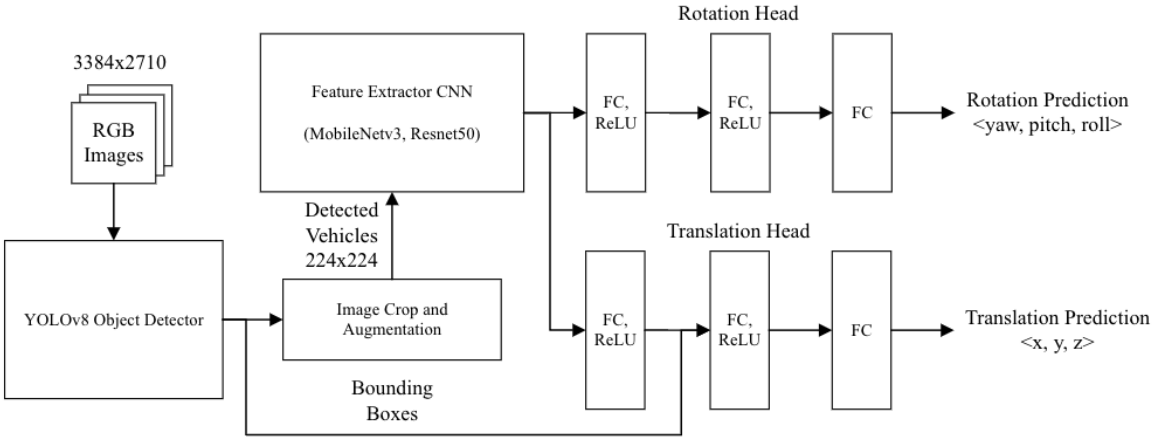

Figure 2. Image Pipeline and Network Architecture of JASMUR

4.2. Data Augmentation

To enhance the robustness of our model, we design a data augmentation step in the JASMUR pipeline using the `albumentations` library [16]. This augmentation is applied to the cropped car images (our region of interest). The augmentation techniques simulate various real-world scenarios and bolster the generalization ability of our model. Our selected augmentation steps are based on the best performing Kaggle entries [17]. The key augmentation techniques we used are listed below, and examples are illustrated in Figure 3.

- **Shift-Scale-Rotate**: Randomly shifts, scales, and rotates the images to simulate different viewpoints.
- **Brightness-Contrast Adjustment**: Randomly adjusts the brightness and contrast of the images to account for different lighting conditions.
- **RGB Shift and Hue-Saturation-Value Adjustment**: Applies random shifts in RGB values and hue-saturation-value to handle color variations.
- **Blurring Techniques**: Uses both Gaussian blur and median blur to mimic the effect of motion and defocus.
- **Channel Shuffle**: Randomly shuffles the color channels to increase the model's robustness to color variations.

4.3. Model Architecture

JASMUR implements a two-head neural network architecture. First, cropped and augmented images are resized and fed in to a pretrained backbone feature extractor CNN

Figure 3. Data Processing Example. Top Left: Original Image. Top Right: YOLO Output. Bottom Left: Cropped Image based on bounding boxes. Bottom Right: Augmented Image.

(ResNet50 [18] or MobileNetV3 [19]) or Vision Transformer (ViT) [20]. Each of these models are pre-trained on ImageNet [21]. MobileNetV3 is relatively lightweight and appropriate for resource-constrained use cases such as embedded systems. ResNet, at the cost of more computation and memory, brings higher accuracy and is able to capture more complex patterns. ViT is effective for advanced image recognition through its transformer architecture but it comes with higher demands on processing power and data.

We remove the classifier layer of the CNN/transformer backbone and feed its learned features to parallel rotation and translation heads. Each head begins with a fully connected layer to capture information from the feature extractor independent of the other head.

Rotation head is comprised of three fully connected layers with ReLU for non-linearity. Effectively, we use a pre-

trained CNN/transformer with a new regression head to estimate the rotation vector (yaw, pitch, roll) and output three Euler angles in radians.

Translation head is also comprised of three fully connected layers with ReLU for non-linearity. Similar to rotation, it first takes in the backbone features. We additionally calculate four features using the bounding box vertices outputted by YOLO; (1,2) bounding box center coordinates x,y, (3) bounding box width and (4) bounding box height. These are normalized to the original input image size, and the center x,y coordinates are zero-centered. We intuit that there is a transformation from the bounding box center point to the vehicle location that is learnable, while the size of the bounding box exposes information about the distance of the vehicle relative to the camera (z coordinate). These features are concatenated with the output of the 1st fully connected layer that learns the CNN/transformer features, and fed in to the 2nd fully connected layer of the translation head. The final layer of each head reduces the features to a single 3D vector, producing three coordinates in normalized image space (x, y, z). The network jointly predicts the 6D pose of the vehicle.

4.4. Loss Functions

We treat 6D pose regression as multi-task learning. We separately regress position and orientation of each vehicle, and define separate loss functions for the two tasks.

Translation Loss: $L_{translation}$ is regressed in pixel units normalized to the original image space for x and y, and pixel units normalized to the maximum seen depth for z. We adopt L1 (Manhattan) distance to keep the loss value scale comparable to rotation loss, as defined in equation 1, where t_i is the (x,y,z) label for a given vehicle, and $\hat{t}_i$ is the (x,y,z) prediction.

$$\mathcal{L}_{translation} = \frac{1}{N}\sum_{i=1}^{N}|t_{ij} - \hat{t}_{ij}| \tag{1}$$

Quaternion conversion: For rotations, top performing entries [1, 17, 22] prefer alternative representations to Euler angles. Euler angles are not one-to-one. The same angle can be represented by multiple values at 2π offsets, which turns into a multi-modal task for regressing it, and presents a challenge. To alleviate this problem, we convert the 3D rotation labels and our output prediction to quaternion vectors $q = (q_0, q_1, q_2, q_3)$ as shown [23].

$$q_0 = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \tag{2}$$

$$q_1 = \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \tag{3}$$

$$q_2 = \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \tag{4}$$

$$q_3 = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) \tag{5}$$

- ϕ is the roll angle (rotation around the x-axis)
- θ is the pitch angle (rotation around the y-axis)
- ψ is the yaw angle (rotation around the z-axis)

Rotation Loss: $L_{rotation}$ is regressed as quaternions using L2 (Mean-Squared-Error) distance as shown in Equation 7. This loss measures the angular difference between predicted and true label rotations in 4D space.

$$\mathbf{q}_{\text{label}} = \frac{\mathbf{q}_{\text{label}}}{\|\mathbf{q}_{\text{label}}\|} \quad \mathbf{q}_{\text{pred}} = \frac{\mathbf{q}_{\text{pred}}}{\|\mathbf{q}_{\text{pred}}\|} \tag{6}$$

$$\mathcal{L}_{quaternion} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{q}_{\text{label},i} - \mathbf{q}_{\text{pred},i}\right)^2 \tag{7}$$

Finally, we combine the translation and rotation losses into one joint loss by simply summing the two terms as shown in equation 8.

$$\mathcal{L}_{joint} = L_{translation} + L_{rotation} \tag{8}$$

Using L1 loss for translation instead of L2 loss has the effect of producing smaller loss numbers in magnitude, which brings the two to the same scale, preventing either term from dominating and causing poor learning for multiple tasks. Note that we do not apply scaling factors for either loss term, but instead define two independent heads and reduce the effect of one task's gradients on the other.

5. Experiments

We first describe our choice of hyperparameters and evaluation metrics, then discuss observations from our experiments. The provided results in Table 3 and Table 4 show the performance metrics of different models (ViT, ResNet50, and MobileNetV3) with various configurations (frozen and unfrozen weights) and the MSE loss function. We provide an analysis of those results, and additional insights from our defined metrics.

5.1. Hyperparameters

We select a mini-batch size of 64 images for training performance and for more accurate batch normalization when fine-tuning the pre-trained backbone networks. Note that since we are using pre-trained backbones, the effect of the minibatch size on batchnorm is muted. We use the Adam optimizer with a learning rate of 1e-4 and betas=(0.9,0.999). This was carefully tuned for learning stability, as larger values caused loss spikes and had trouble converging. Additionally, we add a learning rate scheduler to reduce the learning rate to a factor of 0.1 every 3 epochs, which helps us take smaller steps towards the end of training.

5.2. Dataset Split and Evaluation Metrics

The provided training dataset is split into two, training and validation. The validation set is randomly selected and represents 20% of the training set. While the competition provides a testing set, ground truth for the images are not provided. Since the competition is no longer accepting submissions for evaluation, we are unable to compare our results with other entries in the competition.

The competition defines a mean average precision (mAP) metric to evaluate differences between the predicted pose information and true position and rotation of each vehicle. We first implement the two distance functions, rotation distance and translation distance as specified by the competition [2], getting the angular difference between two quaternions in degrees and getting the root-sum-square (RSS) distance between the two spatial locations.

The distances are then compared to each step in the threshold list shown in Table 2. If both distances are within the tolerance within a given step, it is considered a true-positive, and at least one mismatch is considered a false-positive. The competition defines rotation tolerances in degrees and translation tolerances in meters. However, we do not have access to the image scale. We re-define the units of translation tolerance to be in normalized pixel space for our evaluation.

Category	Thresholds									
Rotation	50	45	40	35	30	25	20	15	10	5
Translation	0.1	0.09	0.08	0.07	0.06	0.05	0.04	0.03	0.02	0.01

Table 2. mAP Thresholds

We use validation mAP as the final evaluation metric due to the nature of the 2020 Kaggle competition, where we can no longer submit results for evaluation.

5.3. Best Performing Model

ResNet50 with frozen weights and MSE loss (`ResNet_freeze`) stands out as the best performing model. This configuration achieves an average validation loss of 0.245 5, 3, an average validation mAP of 0.258, an average validation rotation mAP of 0.446, an average validation xy mAP of 0.999, and an average validation xyz mAP of 0.467 4. The drop in mAP between xy and xyz is due to our normalization of the Z value to the maximum value instead of making an actual prediction, which means only the prediction on xy makes sense. While the average validation loss is lower for ResNet with unfrozen weights, mAP for both translation and rotation (`Avg Val All mAP`) is higher for ResNet50 with frozen weights. It's possible to have a higher loss but still achieve high precision metrics because the loss function penalizes all errors, while the precision metric mAP rewards on the accuracy of predictions for rotation and translation.

5.4. Model Performance

ResNet50 consistently outperforms other models 5, 4, 5, 6. MobileNetV3 with unfrozen weights and MSE loss also performs well, but slightly behind ResNet50 in most metrics. On the other hand, ViTs generally perform worse compared to ResNet50 and MobileNetV3.

5.4.1 Model Comparison

ViTs need large datasets because they lack the inherent biases of CNNs like locality and translation invariance. Although ViTs are great at capturing global dependencies, they struggle with fine-grained details without substantial data and resources and therefore even with pre-trained ViTs, their performance on car pose estimation with Kaggle data may be limited. On the other hand, CNNs like ResNet and MobileNet, which are also pre-trained on large-scale datasets such as ImageNet, excel at capturing local features and building hierarchical representations, crucial for 3D pose estimation. Pre-trained ResNet and MobileNet models have proven effective in various tasks with large-scale datasets. ResNet performs better than MobileNet due to its deeper architecture, which captures complex features needed for detailed 3D pose estimation, while MobileNet's design, optimized for efficiency, may miss intricate details.

5.4.2 Effect of Freezing Weights

Freezing weights bring several advantages. It can prevent overfitting smaller datasets like ours and brings consistent performance to training. A key benefit is that it prevents two independently learned tasks from interfering with each other. When the two regression heads merge at the feature extractor backbone, conflicting gradient directions from the losses can cause one loss term to dominate, hindering one of the tasks from learning effectively. We experienced this problem with an earlier network architecture that branched from a common fully connected layer. With the architecture in 2 and frozen networks, this risk is eliminated.

Name	Train			Validation		
	Total Train Loss	Training Translation Loss	Training Rotation Loss	Avg Val Loss	Avg Val Translation Loss	Avg Val Rotation Loss
ViT_freeze	0.239	0.008	0.238	0.249	0.023	0.227
MobileNet_freeze	0.240	0.009	0.240	0.249	0.023	0.225
ResNet_freeze	0.237	0.007	0.236	0.245	0.021	0.224
MobileNet	0.218	0.008	0.219	0.234	0.022	0.212
ResNet	0.216	0.008	0.216	0.237	0.022	0.215

Table 3. Performance metrics for various model configurations in terms of training and validation losses.

Name	Avg Val All mAP	Avg Val Rot xy mAP	Avg Val Rot mAP	Avg Val xy mAP	Avg Val xyz mAP
ViT_freeze	0.215	0.371	0.371	0.994	0.434
MobileNet_freeze	0.227	0.395	0.396	0.974	0.439
ResNet_freeze	0.258	0.446	0.446	0.999	0.467
MobileNet	0.255	0.462	0.463	0.994	0.451
ResNet	0.249	0.451	0.451	0.998	0.450

Table 4. Performance metrics for various model configurations in terms of mean Average Precision (mAP).

However, unfreezing and fine-tuning the feature extractors presents flexibility and better adaptation to our problem domain. We observed that ResNet consistently performed well, but MobileNet benefited from being frozen as seen in Table 4,6. MobileNet, optimized for efficiency and lightweight design, likely trains better with unfrozen weights. Nonetheless, there is room for improvement by exploring the optimal number of layers to unfreeze and fine-tune while leaving others frozen.

5.5. Areas to Improve

1. **Better Data Augmentation:** Enhancing data augmentation techniques can help the model generalize better to unseen data. One particular area to explore is to reduce the size gap between small and large cars or flip images to make the model more robust.

2. **Post-processing of Images:** Implementing post-processing techniques to remove irrelevant images can improve the model's accuracy; however, we did not have enough time to validate all the detected objects from images. Filtering out images that do not contribute to the task can help focus the training process on more relevant data, but this requires many human hours to implement.

3. **Precision in x, y, z Predictions:** The precision in x, y predictions is significantly higher compared to including z. This is because our current predictions are primarily focused on x, y, and the z component is not computed as accurately. Improving the computation of the z component can lead to better overall predictions. We only exposed box width and height as a feature that qualitatively could tell us information about z—smaller box means farther away. Since a single RGB image does not contain much depth information, we normalized the z labels to the maximum z value seen and regressed that way. We found out that a single image simply does not carry enough depth information, limiting our precision, as it did for every top entry in the competition.

4. **Inference** Testing our model on unseen real world data. The competition does not provide labels for the test set, so we do not have a clear way to evaluate our model with a test set and compare our performance against the other entries.

5. **Loss functions** We were unable to achieve a low loss value for rotation prediction, which impacted the overall loss. Experimenting with advanced loss functions may better capture the nuances of rotational prediction.

6. Conclusion

ResNet50 with frozen weights is the best-performing configuration for this specific task, demonstrating strong generalization and adaptability. Vision Transformers may not be as suitable due to their data efficiency and complexity issues. Frozen weights provide stability and efficiency, particularly in transfer learning scenarios, although they may limit the model's ability to adapt fully to new data. The additional insights from averaged metrics further support these conclusions, highlighting the superiority of ResNet and the advantages of using unfrozen weights.

Acknowledgements

We would like to thank David Rubinstein (Spotify) for his invaluable input and assistance. Special thanks to the Kaggle community for providing the Peking University/Baidu Autonomous Driving dataset and for their insightful discussions and shared resources. We also appreciate the developers of the YOLOv8 and `albumentations` libraries for their powerful tools that significantly contributed to the success of this project. Finally, we acknowledge the support and resources provided by our institution, which made this research possible.

Model	Avg Val Loss	Avg Val Translation Loss	Avg Val Rotation Loss	Avg Val All mAP	Avg Val xy mAP	Avg Val Rot mAP
Mobilenet	0.242	0.023	0.219	0.233	0.985	0.425
Resnet	0.241	0.022	0.219	0.245	0.998	0.454
Vit	0.248	0.023	0.225	0.217	0.984	0.380

Table 5. Performance metrics averaged across models.

Frozen Weights	Avg Val Loss	Avg Val Translation Loss	Avg Val Rotation Loss	Avg Val All mAP	Avg Val Rot mAP	Avg Val xy mAP
No	0.238	0.023	0.215	0.243	0.460	0.995
Yes	0.247	0.023	0.224	0.229	0.406	0.987

Table 6. Performance metrics averaged across frozen and unfrozen weights.

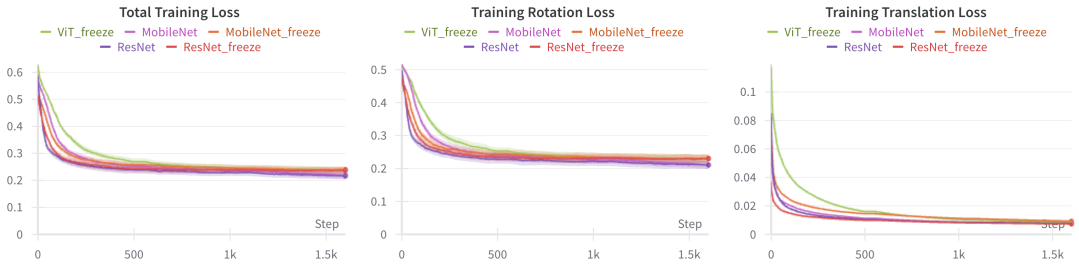

Figure 4. Training Loss

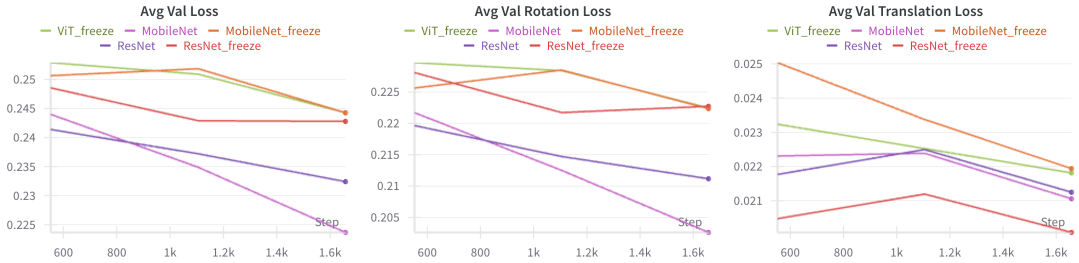

Figure 5. Validation Loss

Figure 6. Validation mAP scores

References

[1] Di Wu, Zhaoyong Zhuang, Canqun Xiang, Wenbin Zou, and Xia Li. 6d-vnet: End-to-end 6dof vehicle pose estimation from monocular rgb images. 06 2019.

[2] Ruigang Yang zdf56 Addison Howard, Phil Culliton. Peking university/baidu - autonomous driving, 2019.

[3] Waymo Satish Jeyachadran. Introducing the 5th-generation waymo driver, 2020.

[4] The Waymo Team. Utilizing key point and pose estimation, 2022.

[5] Jingxiao Zheng, Xinwei Shi, Alexander Gorban, Junhua Mao, Yang Song, Charles R. Qi, Ting Liu, Visesh Chari, Andre Cornman, Yin Zhou, Congcong Li, and Dragomir Anguelov. Multi-modal 3d human pose estimation with 2d weak supervision in autonomous driving, 2021.

[6] Lei Ke, Shichao Li, Yanan Sun, Yu-Wing Tai, and Chi-Keung Tang. Gsnet: Joint vehicle pose and shape reconstruction with geometrical and scene-aware supervision. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV*, page 515–532, Berlin, Heidelberg, 2020. Springer-Verlag.

[7] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection, 2019.

[8] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-yuan Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 07 2022.

[9] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8, 2024.

[10] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection, 2024.

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[13] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, David Weiss, Ben Sapp, Zhifeng Chen, and Jonathon Shlens. Scene transformer: A unified architecture for predicting multiple agent trajectories. *arXiv*, 2021.

[14] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects, 2024.

[15] Xibin Song, Peng Wang, Dingfu Zhou, Rui Zhu, Chenye Guan, Yuchao Dai, Hao Su, Hongdong Li, and Ruigang Yang. Apollocar3d: A large 3d car instance understanding benchmark for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5452–5462, 2019.

[16] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2):125, February 2020.

[17] 4uiiurz1. 5th place entry on kaggle pku autonomous driving. `https://github.com/4uiiurz1/kaggle-pku-autonomous-driving`, 2019.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[22] Kaggle Top 2nd entry. Pku autonomous driving discussion. `https://www.kaggle.com/competitions/pku-autonomous-driving/discussion/127099`, 2020. Accessed: 2024-06-04.

[23] Wikipedia. Conversion between quaternions and euler angles, 2024.