Generating Top-Down View from 6 Stereo Images

Muhammad Osama Khan [1] **Muhammad Muneeb Afzal** [1] **Divya Juneja** [1]

Abstract

Team 19 achieved best threat scores of 0.762 and 0.0143 on the validation set on the road segmentation and object detection tasks respectively. Furthermore, we ranked 8th overall on the leaderboard.

1. Introduction and Related Works

Top-down view estimation involves 2 main components – road segmentation and object detection. Road segmentation falls under the realm of semantic segmentation. Some noteworthy works in this domain include SegNet, which stores the max-pooling indices in the encoder layers to upsample (max unpool) during the decoding step, thereby eliminating the need for *learning* to upsample. Other key works include (Long et al., 2015) which introduced fully convolutional networks using deconvolution (transposed convolution) layers for semantic segmentation.

In the realm of 2D object detection, YOLO v3 (Redmon & Farhadi, 2018) and Faster-RCNN (Ren et al., 2015) are some state-of-the-art works. YOLO v3 divides the image into grids and detects several bounding boxes per grid whereas Faster-RCNN generates region proposals which are then used for detection.

Self-supervised learning is used to learn feature representations from unlabeled data. Particularly, pretext tasks such as Jigsaw (Noroozi & Favaro, 2016) have proved to work well for learning feature representations for downstream vision tasks. In this work, we take inspiration from the aforementioned semantic segmentation, object detection and self-supervised methods and introduce novel features specific to the task at hand.

2. Road Segmentation

We designed our own architecture for road segmentation. The final architecture that we submitted for the competition consists of 6 VGG-like encoders with shared weights and a single decoder (see Figure 1). Our main novelty is in designing pretext tasks specifically suited to this architecture (see sections 2.2 and 2.3).

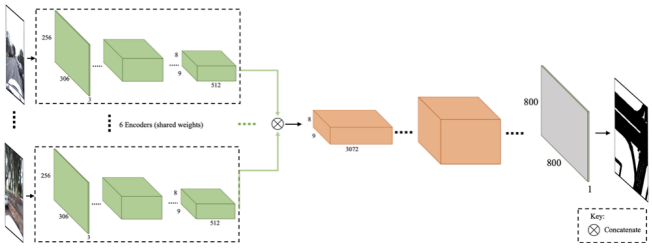

Figure 1. Road Segmentation Architecture

2.1. Architectures

2.1.1. Combining 6 Intermediate Feature Maps

Three strategies were tried for combining the 6 feature maps produced by the 6 encoders: concatenation, mean and weighted average. Concatenation produced the best results (see Table 1).

Table 1. Combining Feature Maps

Mean	Weighted Average	Concatenate
0.681	0.685	*0.741*

2.1.2. Decoder Architectures

For designing decoder architectures, we experimented with 2 strategies. Firstly, we used up-sampling and 2D convolutions to increase the spatial representations whereas, secondly, we used transposed convolutions. We observed that the latter yielded better results as shown in Table 2.

Table 2. Decoder Architectures

Upsampling + Convolutions	Transposed Convolutions
0.733	*0.741*

2.2. Using Extra Information (Lane Masks)

Our team leveraged the extra information provided to us in the form of lane masks in 3 ways. Firstly, we trained the 6 encoder network for lane segmentation and then transferred

the weights for road segmentation. This achieved a lower threat score than no pretrain. Secondly, we designed a 6 encoders, 2 decoders architecture to predict both road and lane masks simultaneously. This approach achieved slightly better results than no pretrain. Thirdly, and perhaps most interestingly, we designed a correspondence pretext task. Given 2 inputs, road mask and lane mask, the network had to predict if the two masks correspond. For instance, the road and lane masks in the left column of Figure 2 correspond since they are taken from the same scene whereas the ones in the right column do not. This idea was inspired from (Arandjelovic & Zisserman, 2017), where the task is to predict if the image and audio correspond. Out of the 3 approaches we tried, the lane-road correspondence approach yielded the best results, although we found it required careful initialization to train effectively (see Table 3).

Table 3. Ablation study of 3 ways of using lane masks

NO PRETRAIN	TRANSFER LEARNING	LANE + ROAD SEGMENTATION	LANE – ROAD CORRESPONDENCE
0.741	0.736	0.743	*0.746*

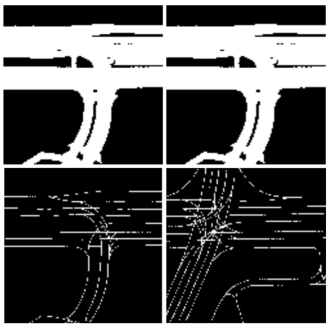

Figure 2. Lane-road correspondence pretext task

2.3. Self-supervised Learning

2.3.1. Pretext Tasks

Next, we utilized self-supervised pretext tasks in order to pretrain the network on the unlabeled dataset. Here, we used 2 pretext tasks: jigsaw and our own designed pretext task which we call ***stereo***. Similar to the jigsaw idea, the task is the following: given a randomly permuted sequence of images from the 6 cameras, the network has to predict the permutation. The intuition behind designing this task was that in order to accurately perform road segmentation for the entire scene, the network needs to know which camera represents which part of the scene. However, the stereo pretrain network could simply cheat by looking at the body of the ego-car that is visible in each image. Since this view of the ego-car is fixed across different frames, the network would not learn anything about the scene itself. In order to avoid this, we crop each of the 6 images to the central 150×150 region, thereby removing any view of the ego-car from the images. In comparison to the jigsaw pretext task, pretraining with the stereo pretext task yielded better results when both methods were pretrained for 700 permutations. However, increasing the jigsaw permutations to 1000 yielded better results (see Table 4).

Note: Stereo has maximum permutations = 6! = 720.

Table 4. Jigsaw and stereo pretext tasks comparison

NO PRETRAIN	JIGSAW (700)*	STEREO (700)*	JIGSAW (1000)*
0.741	0.750	0.753	*0.762*

*represents number of permutations

2.3.2. Transfer Learning

While pretraining with the jigsaw and stereo pretext tasks, note that only the road segmentation *encoder* (see Figure 1) was trained. For transfer learning, we experimented with 3 schemes – 1) Training only the decoder and keeping the encoder weights fixed from the pretrain task, 2) Training both encoder and decoder with the same learning rate, and 3) Training decoder with a higher learning rate than the encoder. The third strategy produced the best results (see Table 5).

Table 5. Transfer Learning

NO PRETRAIN	DECODER ONLY	ENCODER + DECODER	ENCODER + DECODER (DIFFERENT LRS)
0.741	0.737	0.756	*0.762*

2.4. Visualizations

- Strengths: The network predicts long and wide roads, especially those located at the center of the scene, very accurately as can be seen from the top row of Figure 3.
- Weaknesses: The network struggles to predict narrow roads, especially those located at the boundaries of the scene, as can be seen from the bottom row of Figure 3. These failures can be attributed to occlusion and the infrequency of these types of roads in the dataset.

In order to verify our hypothesis that the stereo pretrain task does indeed help the network learn the different camera views, we did a T-SNE visualization of the encoded features from the stereo pretrain network. Figure 4 shows that the network clusters the feature representations for the 6 camera

images in different regions of the latent space, thereby helping the network perform the road segmentation task since it can distinguish between the different camera views and match each camera view to a particular region of the scene road segmentation result.

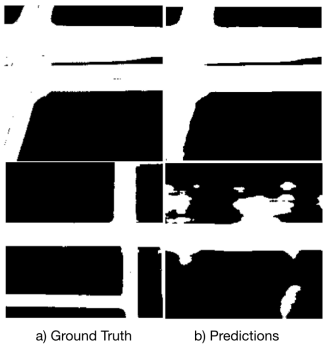

Figure 3. Positive and negative examples

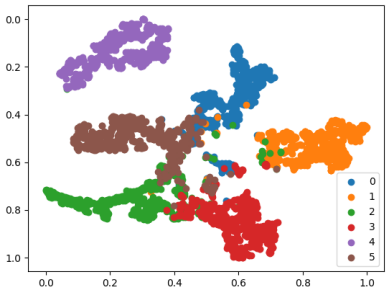

Figure 4. T-SNE visualization of encoded features from stereo pretrain network. The axes have been normalized to 0-1. The legend represents the 6 camera views 0-5.

3. Object Detection

For object detection, we experimented with two types of architectures: Modified YOLOv3 (Redmon & Farhadi, 2018) and Faster R-CNN (Ren et al., 2015).

3.1. Modified YOLOv3

3.1.1. Input Types

Firstly, we experimented with different types of inputs to our network and chose the best way to input our data. The two different types of inputs used are shown in Figure 5.

Firstly, we resized the original images of sizes 256 × 256 × 3 to 288 × 288 × 3 and then concatenated them to obtain an input feature map of size 288 × 288 × 18. Secondly, we tiled the 6 input images into a single image of dimension 512 × 918 × 3 and then resized the image to 800 × 800 × 3. Note that 3 images corresponding to the front view of the car were tiled in the first row whereas the 3 images corresponding to the back view of the car were tiled in the second row as shown in Figure 5.

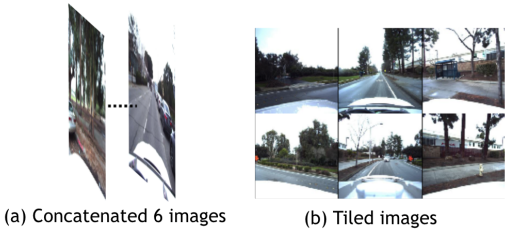

Figure 5. Input type for our objection detection network

Table 6 shows that we obtained superior results in the case of tiled images. We hypothesize that the network was able to learn the top-view from the tiled images since it was able to reason from 3 images at the front and 3 images at the back. Hence it was able to reconstruct the top-view and finally predict the required bounding boxes. However, for concatenated images, it was very difficult to learn the spatial relationship between the 6 images and reconstruct the top-view to be able to predict the bounding boxes.

Table 6. Input Types

CONCATENATED	TILED
0.000143	*0.000498*

3.1.2. Different Architectures

We experimented with different types of architectures with different preprocessing of the targets for bounding boxes.

Vanilla YOLOv3 with Single Targets The main difficulty in using YOLO-like architecture was that it accepts inputs in the form: $x_{centre}, y_{centre}, width, height$. However, in our given task, we are given 4 coordinates of the bounding boxes (i.e. 8 values). In vanilla architecture, we simply converted the 4 coordinates to the required $x_{centre}, y_{centre}, width, height$ by using the formulae given below. Note that in this case, we assume that the boxes are not tilted. Since we did not get as great results as we expected (See Table 7), we realized that we needed to incorporate tilted boxes in our network design. This led to our proposed *Modified YOLOv3: Double Targets* architecture.

Given original inputs (x_i's, y_i's), we do the following preprocessing:

$$x_i = 10x_i + 400; \quad y_i = 10y_i + 400, \quad \forall i = 1, .., 4$$
$$x_{centre} = \frac{(x_1 + x_4)}{2}/800; \quad y_{centre} = \frac{(y_1 + y_4)}{2}/800$$
$$width = |x_2 - x_3|/800; \quad height = |y_2 - y_3|/800$$

Table 7. Different Architectures for YOLOv3

VANILLA	DOUBLE TARGETS
0.000297	*0.000498*

Modified YOLOv3:Double Targets If we assume that the bounding boxes are not tilted, it means we are under-utilizing our targets and thus losing essential information. To rectify this problem, we introduced the idea of double targets, which means that not only did we input the four values $x_{centre}, y_{centre}, w, h$, but we also input extra targets (original 8 values) of the target. The pipeline for Double Targets is shown in Figure 6.

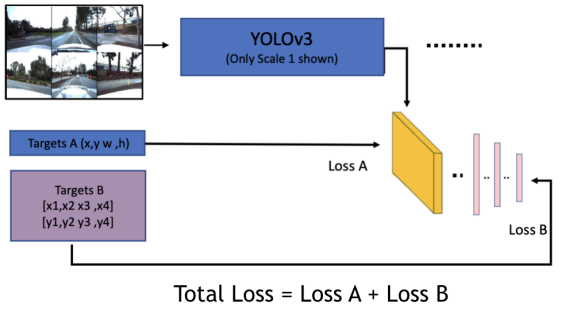

Figure 6. Double Targets architecture - only one head (scale) of YOLOv3 is shown

In order to incorporate tilted boxes, which means predicting 8 coordinates, we added a few linear layers after the 4 coordinates output produced by YOLO. As shown in Figure 6, we compare Targets A with 4 outputs and Targets B with output from linear layers, thus giving us Loss A and Loss B. By using double targets, we improved our performance as illustrated in Table 7.

3.2. Modified Faster R-CNN

Although we tried various ideas for YOLOv3, we obtained best results for object detection with modified F-RCNN.

3.2.1. DIFFERENT LOSS FUNCTION

We modified the loss of the original Faster R-CNN by giving different weights to different parts of the loss function.

Table 8. Loss function weights

COORDINATES (REGRESSION)	OBJECTNESS	CLASSIFIER	RPN REGRESSION	SCORE
1.0	1.0	1.0	1.0	0.0096
5.0	1.0	1.0	1.0	0.0143

Note that our task is class agnostic i.e. we do not care which class the bounding box belongs to. Therefore, in order to utilize this laxation, we weighted the four parts of the loss functions in order to obtain an optimized loss function for our use case. As per Table 8, increasing the weight for regression loss yields the best result.

Strengths and Weaknesses of YOLOv3 and FRCNN We note that the performance of modified FRCNN was far superior than the modified YOLOv3 architecture. According to our hypothesis, this is because YOLOv3, with its mechanism of predicting anchors in each of the grid cells, is too attuned for accepting single images and predicting bounding boxes for one image. In other words, YOLOv3, by dividing the input images into grid cells, tries to find bounding boxes in each of the grid cells and thus relies on local features of one image. This means that upon receiving 6 images, it is not able to reason about bounding boxes across images properly. However, FRCNN architecture with its region proposals idea is more robust to these issues than YOLOv3.

4. Conclusion

In summary, we obtained our best road segmentation threat score of 0.762 when using the 6 encoders siamese architecture along with the jigsaw and stereo pretext tasks for pretraining. For object detection, we obtained our best threat score of 0.0143 when using a modified version of Faster RCNN. We were placed 8th overall on the leaderboard.

References

Arandjelovic, R. and Zisserman, A. Objects that sound. *CoRR*, abs/1712.06651, 2017. URL http://arxiv.org/abs/1712.06651.

Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

Noroozi, M. and Favaro, P. Unsupervised learning of visual representations by solving jigsaw puzzles. *CoRR*, abs/1603.09246, 2016. URL http://arxiv.org/abs/1603.09246.

Redmon, J. and Farhadi, A. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 91–99. Curran Associates, Inc., 2015. URL https://arxiv.org/pdf/1506.01497.pdf.