Supplementary
Binary TTC: A Temporal Geofence for Autonomous Navigation

1. Additional Details on Architecture

1.1. Binary Estimation

Our binary segmentation architecture has three main components: a feature extraction network (FeatExtractNet), a segmentation network (SegNet2D), and a refinement network (SegRefineNet). Our architecture blocks are adopted from Bi3DNet [1]. We do not use any batch normalization layers in our network.

FeatExtractNet. We use the same feature extraction network as used in Bi3DNet [1]. This feature extraction module is based on the simplified version of the feature extraction network from PSMNet [2]. We normalize each color channel of the input images using the mean and standard deviation of 0.5 before passing to this network. The output is a 32-channel feature map at one third of the input image resolution.

SegNet2D. We warp the source image features using a task-dependent operator. To avoid cropping out the image features during this warping operation, we zero pad the feature maps to $1.5\times$ the feature map resolution. The reference image feature map and the warped source image feature map are concatenated and fed to the SegNet2D network. SegNet2D is a 2D encoder-decoder network with skip-connections. The encoder is comprised of five blocks, each of which has a conv layer that downsamples the features with a stride of 2 followed by another conv layer with a stride of 1. We use 3×3 kernels. The feature sizes for each of these five blocks are 128, 256, 512, 1024, and 1024. The decoder is comprised of five blocks, each of which has of a deconv layer with 4×4 kernels and a stride of 2, followed by a conv layer with 3×3 kernels and a stride of 1. The feature sizes for each of these five blocks are 1024, 512, 256, 128, and 64. We use the LeakyReLU activation function in the network with a slope of 0.1. A final conv layer with 3×3 kernels, without any activation, is used to generate 3 outputs: one for binary TTC, one for binary horizontal optical flow, and one for binary vertical optical flow. The final segmentation probability maps can be obtained by cropping out the excess padding, upsampling the outputs to the input image resolution and applying a sigmoid function.

SegRefineNet. SegRefineNet is used to refine the segmentation outputs from SegNet2D network using the reference image as a guide. First, we generate a 16 channel feature map for the reference image by applying 3 conv layers with 3×3 kernels and a stride of 1. The first two layers use ReLU activation and the final layer does not use any activation. This feature extraction is done only once and can be used for refining multiple segmentation outputs from the SegNet2D network. An upsampled segmentation output is concatenated with the reference image features and refined by applying 4 conv layers. Each conv layer uses 3×3 kernels, a feature-size of 8, and LeakyReLU activation with a slope of 0.1. The final output of this network is generated by a final conv layer with 3×3 kernel and without any activation. The final binary segmentation probability map can then be generated by applying a sigmoid function.

1.2. Continuous Estimation

Given input images we generate the corresponding feature maps using the same FeatExtractNet as in Section 1.1. To generate continuous estimation results we uniformly sample the warping parameters in the desired range and use these parameters to warp the features of the source image. These warped source image features are concatenated with the reference image features to form an input volume for the SegNet2D network. SegNet2D generates an output volume which upon upsampling to the input image resolution, applying the sigmoid operator followed by AUC operator gives us the continuous estimation map. This continuous map is further refined using the input image as a guide using a refinement network, ContRefineNet. This network is based on the disparity refinement network proposed in StereoNet [5].

2. Additional Details on Training

2.1. Binary Estimation

We pre-train our network for the binary optical flow task on the FlyingChairs2 [3, 4] dataset for 300 epochs and train it further on FlyingThings3D dataset [6] for 400 epochs using a binary cross-entropy (BCE) loss with respect to a thresholded version of the ground truth. Each batch for training is formed by randomly sampling 16 image pairs from the dataset and then randomly sampling shifts for the

two components of the optical flow vector for each image. Note that our SegNet2D network has three output heads and we leave the head corresponding to binary TTC untouched during this training. We then fine-tune our network for estimating binary TTC first on the Driving [6] for 500 epochs and then on the KITTI15 [7] datasets for 10k epochs. Since both datasets also offer optical flow data, in this second stage we train for both binary optical flow and binary TCC. To do this we form a batch by randomly sampling 16 image pairs from the dataset. Then for each image pair we form two sets of warped image features: one by randomly sampling shifts for the two components of the optical flow vector and other by randomly sampling scales for training binary TTC. We select the appropriate segmentation output corresponding to the task and use BCE loss to train our network simultaneously for the binary optical flow and TTC estimation task. Throughout our training we randomly sample shifts in the range $[-99, 99]$, and scales in the range of $[0.5, 1.3]$. We use relative weights of 0.8 and 0.2 for binary TTC and binary optical flow task respectively. For training on the KITTI15 dataset, we split our dataset into training (160 examples) and validation (40 examples) sets, following the split provided by Yang and Ramanan [8].

2.2. Continuous Estimation

To train our network for the continuous estimation task, we start with the network trained on the FlyingChairs2 and FlyingThings3D datasets for the binary optical flow task and fine-tune it on FlyingThings3D for continuous optical flow for 100 epochs. Each batch for training is formed by randomly sampling 8 image pairs from the dataset. For each training image pair we uniformly sample horizontal and vertical shifts and stack the maps corresponding to each shift. We use the resulting volumes to compute continuous optical flow via the AUC operation which we refine using ContRefineNet. To fine-tune our network for continuous TTC estimation, we follow a similar strategy as for the binary segmentation training. We continue training the network for the task of continuous optical flow and continuous TTC estimation on Driving dataset for 100 epochs, followed by KITTI15 datasets for 600 epochs. We form three volumes, two corresponding for optical flow and one for TTC. Throughout the training, we use BCE loss on the estimated binary segmentation probability maps, and a SmoothL1 (SL1) regression loss on the output of the AUC module. We use relative weights of 0.1 and 0.9 respectively. While training on both TTC and OF estimation tasks, we use relative weights of 0.8 and 0.2 respectively. For optical flow we randomly sample a contiguous block of 16 shifts that are divisible by 3 and in the range $[-99, 99]$ for both the components of the optical flow. We select the shifts to be a factor of 3 since we perform the shifting on the feature maps that are one third the input image resolution. The AUC operation seamlessly handles the objects with shifts that lie beyond sampled range. For TTC estimation, we work in the inverse TTC domain and uniformly sample 24 scales in the range $[0.5, 1.3]$. We perform the training on the cropped images of size 384×576. The cropping is done after the feature warping step. Again for the KITTI15 dataset, we split the dataset into train and validation sets and use the validation set to compare our method with competing approaches. The network trained on the entire KITTI15 dataset is used for scene flow estimation on KITTI15 benchmark images as explained in the main paper.

References

[1] Abhishek Badki, Alejandro Troccoli, Kihwan Kim, Jan Kautz, Pradeep Sen, and Orazio Gallo. Bi3D: Stereo depth estimation via binary classifications. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[2] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[4] Eddy Ilg, Tonmoy Saikia, Margret Keuper, and Thomas Brox. Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In *European Conference on Computer Vision (ECCV)*, 2018.

[5] Sameh Khamis, Sean Fanello, Christoph Rhemann, Adarsh Kowdle, Julien Valentin, and Shahram Izadi. StereoNet: Guided hierarchical refinement for real-time edge-aware depth prediction. In *European Conference on Computer Vision (ECCV)*, 2018.

[6] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[8] Gengshan Yang and Deva Ramanan. Upgrading optical flow to 3D scene flow through optical expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.