TPE-Net: Track Point Extraction and Association Network for Rail Path Proposal Generation

Mohammadjavad Ghorbanalivakili*,1, Jungwon Kang*,1, Gunho Sohn[1], David Beach[2] and Veronica Marin[2]

2023 IEEE 19th International Conference on Automation Science and Engineering (CASE) | 979-8-3503-2069-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/CASE56687.2023.10260541

***Abstract*— One essential feature of an autonomous train is minimizing collision risks with third-party objects. To estimate the risk, the train must be able to identify topological information of all the rail routes ahead on which the train can possibly move, especially within merging or diverging rails. This way, the train can figure out the status of potential obstacles with respect to its route, and hence, make a timely decision. Numerous studies have successfully extracted all rail tracks without recognizing separate rail instances. Still, some image-based methods have employed hard-coded prior knowledge of railway structure and 3D data to associate left-right rails and generate rail route instances. However, we propose a rail path extraction pipeline in which left-right rail pixels of the routes are extracted and associated through a fully convolutional encoder-decoder architecture called TPE-Net. Two different regression branches for TPE-Net are proposed and trained separately to estimate the coordinates of center points of each rail route, along with their corresponding left-right pixels. Extracted rail pixels are then spatially clustered to provide topological information of all the possible train routes (ego-paths), discarding non-ego-path ones. Comparing different TPE-Net baselines and regression branch designs reveals our highest true-positive-pixel level average precision and recall of 0.9290 and 0.8864, respectively, at around 12 frames per second on a challenging, publicly released benchmark. The results also indicate that based on our proposed model selection method, the TPE-Net can reliably understand rail scene configuration and provide useful geometrical information for the autonomous train systems.**

I. INTRODUCTION

With the development of intelligent technology, modern railway system has gradually changed from human driving mode to an autonomous and unmanned mode [1]. One critical feature of an autonomous train system is to avoid collision with potential obstacles [2]. Through such unexpected circumstances, the autonomous train needs to identify its route among all the rail routes ahead and precisely localize obstacles with respect to the detected route. This way, the train can estimate the risk and react accordingly.

Railway structure might occasionally incorporate switch states such as merging or diverging routes. In order to detect the train route among all the candidates, it is essential to localize the switch states and identify their configurations.

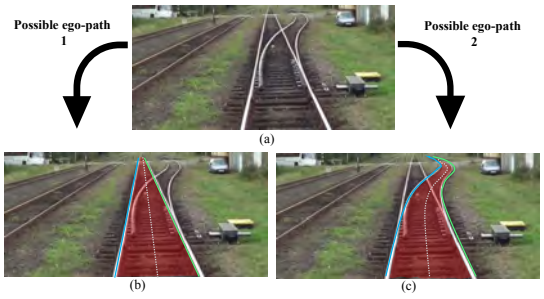

Fig. 1. (a) is a sample input image. (b) and (c) are the corresponding identified possible ego-paths. Other rail routes seen in the input image are discarded as they are not possible ego-paths.

Yet, in this study, we focus on discovering all the possible ego-paths (possible train routes) ahead rather than struggling to find the train's unique route by investigating the switches.

In this context, a *track* is defined as a pair of a left rail and a right rail. Here, we define *track points* as track center points located at the center between a left rail and a right rail. *Rail area* is also defined as the region on the ground surrounded by a track. Fig. 1 illustrates the primary goal of this paper, i.e., identifying all the possible ego-paths along with their corresponding track points, left rail, and right rail pixels in an input RGB image captured by a forward-looking camera installed on a train.

High inference speed, robustness, and the ability to learn both low-level and high-level semantic information have recently made deep neural networks an effective tool for railway object detection, track extraction, switch state recognition, and track inspection [4,5,6]. In [7], a multi-task network uses Mask-RCNN with ResNet101 as its backbone to detect rail areas and estimate collision risk in single-track scenes. In [8], DFF-Net with VGG16 as the backbone performs real-time object and rail area detection. Track and rail area segmentation are also accomplished through CNNs in various studies within different single-track and multi-track scenes [1,3,6,9,10,11]. The aforementioned works have tried to extract all the rail areas or tracks without indicating their instances. Another line of research [12], however, takes a step forward by segmenting tracks of complicated structures using ERF-Net, and then, associating left-right rails to form track instances based on topological features. Another level of railway scene understanding [2] is also achieved through segmenting all rail areas in an image using RailCNN, and then, specifying the possible ego-paths among the segmented regions. Yet, the methodology does not make a distinction

*These two authors are co-first authors, with equal contribution

[1]M. Ghorbanalivakili, J. Kang, and G. Sohn are with the Department of Earth and Space Science and Engineering, Lassonde School of Engineering, York University, 4700 Keele Street, Toronto, Ontario M3J 1P3, Canada mvakili@yorku.ca, kctown99@gmail.com, gsohn@yorku.ca

[2]D. Beach and V. Marin are with Thales Group, Canada David.Beach@thalesgroup.com, Veronica.Marin@thalesgroup.com

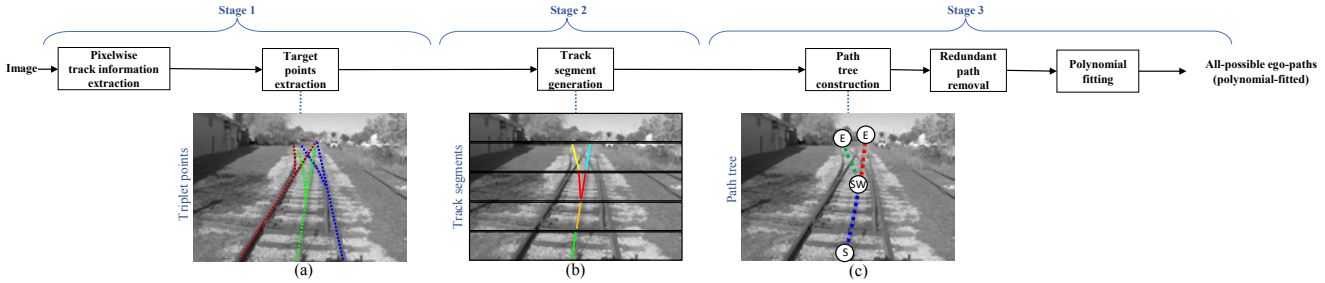

Fig. 2. Overall diagram of our proposed rail path extraction algorithm. (a) shows the first stage through which left, right, and center rail pixels (triplets) are detected by the network. Next, track segments are generated in each subregion by associating the extracted track points as illustrated in (b). Here, segments with the same color fall into the same cluster. Finally, a path tree is created which covers all the possible ego-paths according to (c). In the path tree, *S* stands for start node, *SW* stands for switch node, and *E* stands for end node of each detected ego-path.

between different rail route instances.

Most existing works extract rails without distinguishing between their separate instances. A few studies, however, associate left and right rails to generate track instances [12]. Such an association is mainly done using the tricky RANSAC and the geometrical features of the rail scene. Such features can be extracted from 3D point cloud or inverse perspective transform with the camera parameters provided. Instead of relying on RANSAC to sample rail points, create rail candidates, pick the best ones among all candidates, and pair the rails to generate track instances, we employ deep neural networks to generate strong hypotheses of left-right rail pixel pairs even within multi-track scenes. To find left-right rail pixel associations, the network looks once into the image without any dependency on the hand crafted prior knowledge of the rail shape, scene topology, and geometrical constraints. Therefore, our main contributions can be summarized as the following:

- We propose a fully convolutional neural network incorporating a segmentation and a regression branch. The regression branch estimates the degree of being a track point for all the pixels in the input image. It also regresses pixel-level horizontal distance to the corresponding left and right rail for the detected track points.
- Through a bottom-up spatial clustering of the track points extracted by the neural network, we construct a path tree containing topological and geometrical features of all the possible ego-paths.
- We train three different network baselines on the exact same regression and segmentation tasks to compare their performance in the overall rail path extraction outcome. As a result, we show that any segmentation baseline can be implemented in our proposed rail path extraction method.

II. Methodology

A. System Overview

Fig. 2 explains the overall structure of the proposed rail path extraction algorithm. To tackle complex structures of rail paths, the algorithm employs a bottom-up process similar to [13] through the following three major stages: At first, the network extracts pixel-level triplets (left, right, and center rail pixels) of all the track instances. Next, track segments are generated by spatially clustering the previously extracted track points. Finally, linking the track segments constructs all the possible ego-path trajectories in the shape of a path tree.

B. Pixel-level Track Points Extraction

Inspired by [14], we model each track instance as a group of corresponding track points. To extract the track points, a deep neural network called *Track Point Extraction Network (TPE-Net)* estimates the degree (probability) of being a track point for all the pixels through a heatmap resulting from regression branch. With the coordinates of the left and right rail pixels for each track instance provided in the dataset, the ground truth of the probability heatmap is generated according to (1).

$$I^{C}_{GT}(x,y) = max(min(d^{(x,y)}_{L_1}, d^{(x,y)}_{R_1})/W^{x}_{1}, ..., min(d^{(x,y)}_{L_n}, d^{(x,y)}_{R_n})/W^{x}_{n}) \quad (1)$$

Here, (x, y) coordinates outside the rail areas take zero ground truth value. $d^{(x,y)}_{L_n}$ and $d^{(x,y)}_{R_n}$ are the horizontal distance from (x, y) to the left and right rail, respectively, within the nth rail area instance. W^{x}_{n} is half the track width (distance between a pair of a left and a right rail) of the nth route in vertical position x of the image frame.

Each track point is considered to be associated with a left rail and a right rail pixel lying on the corresponding track instance, all together forming a triplet. Thus, a track instance incorporates a cluster of triplets. To extract the corresponding left and right rail pixel of each detected track point, the network is supposed to estimate pixel-level horizontal distance of a track point to its left and right rail within the image frame. Each distance can be estimated through a heatmap as well. Therefore, the network projects an input image into three single-channel heatmaps, all having the same resolution as the input. The ground truths of the distance-to-right and distance-to-left heatmaps are generated according to (2) and (3), respectively. Similar to (1), (x, y) coordinates outside the rail areas take zero value. Fig. 3 illustrates an overview

Fig. 3. Expected regression outcomes of the first design of the regression branch in a sample rail area of the input image. Each outcome is a 1-channel heatmap. Here, the heatmap value gets larger as the color goes darker.

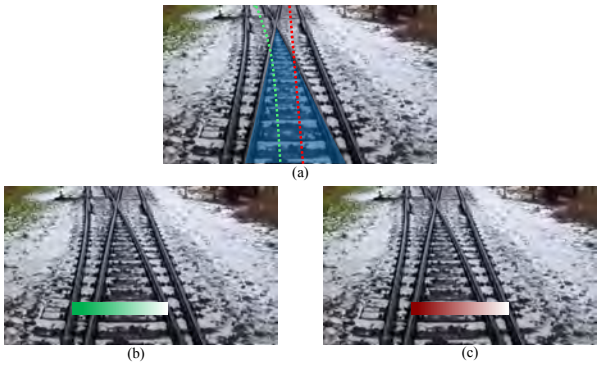

Fig. 4. (a) is a rail scene with two overlapping routes and their corresponding track points. Here, the area shared between the two is annotated in blue. (b) and (c) are the expected distance-to-right heatmaps for the leftmost and the rightmost route respectively, within a sample rail area of each route.

of the expected outputs of the network's regression branch.

$$I^{R}_{GT}(x,y) = max(d^{(x,y)}_{R_1}, ..., d^{(x,y)}_{R_n}) \tag{2}$$

$$I^{L}_{GT}(x,y) = max(d^{(x,y)}_{L_1}, ..., d^{(x,y)}_{L_n}) \tag{3}$$

Coordinates of the peaks in the probability heatmap are found by applying nonmaximum suppression (NMS) within each row, resulting in x-y coordinates of the track points. To find the right rail pixel of an extracted track point, we pick the value of the distance-to-right heatmap in the corresponding track point's x-y coordinates. We employ the exact same procedure to detect the left rail pixel. Note that to this end, a track point is associated with its right and left points, generating a triplet. However, there is no association between the extracted triplets.

According to Fig. 4, triplet point extraction would be occasionally inaccurate in rail switches. Here, there is more than one distance associated with each pixel in the shared rail area, making it impossible to define single-channel ground truth distance heatmaps. The aforementioned version of the regression branch is named *3-channel* regression (i.e., probability, distance-to-left, and distance-to-right heatmaps). To overcome the shortcoming, we propose the latest version of regression branch, in which the probability heatmap is modified to regress two types of attributes at once, i.e., the degree of being a track point and the equal horizontal distance to the left rail and right rail, for each extracted track point. So, this newer regression branch design outputs a single channel heatmap having ground truth values according to (4). Fig. 5 shows a rail scene with its ground truth

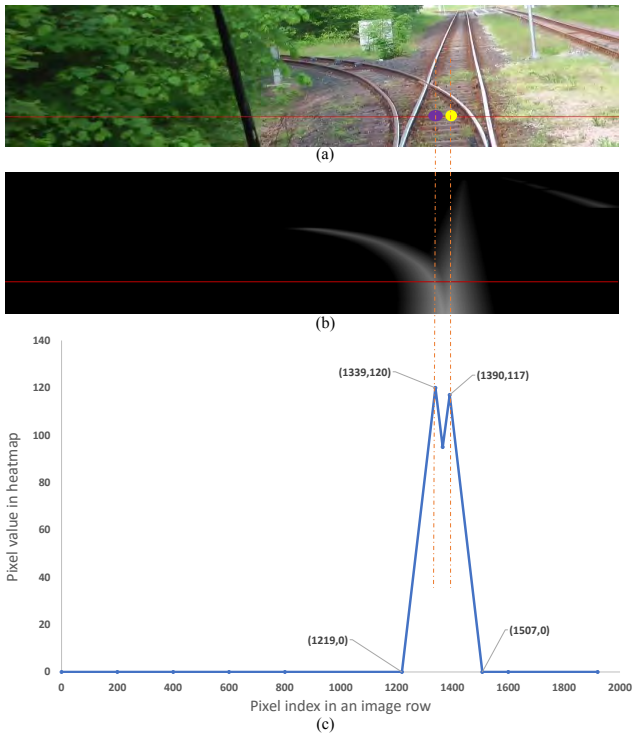

Fig. 5. (a) is a rail scene with two overlapping routes and their corresponding track points within a sample image row. (b) is the corresponding 1-channel regression heatmap ground truth. (c) is a graph showing the pixel values of the ground truth regression heatmap along the selected image row.

regression heatmap and a graph of the heatmap pixel values in a sample image row. Coordinates of the graph peaks give the track point locations, while the peak values estimate the equal distance to the left or right rail for each detected track point. We argue that this approach performs more reliably throughout the switches compared to 3-channel regression. Such a claim is due to the fact that each track point in practice is always individual to only one track instance, so the track points are never overlapping. Also, a track point has ideally an equal distance to its left rail and right rail.

$$I_{GT}(x,y) = \\ max(min(d^{(x,y)}_{L_1}, d^{(x,y)}_{R_1}), ..., min(d^{(x,y)}_{L_n}, d^{(x,y)}_{R_n})) \tag{4}$$

Inspired by the great success of recent multi-task networks [15], we use multi-task learning to leverage semantic labeling in generating our desired regression outcomes. To this end, semantic segmentation is not our main task in order to generate rail path proposals. Yet, introducing segmentation beside regression branch boosts network generalization by exploiting the domain-specific information obtained by the segmentation branch. This way, a shared representation of semantic and geometric features of the track instances is achieved.

For the TPE-Net, any off-the-shelf semantic segmentation network can be employed by simply adding one of our proposed regression branch designs on top of the main segmentation branch of the original network. Yet, to obtain the preliminary path extraction results based on our proposed

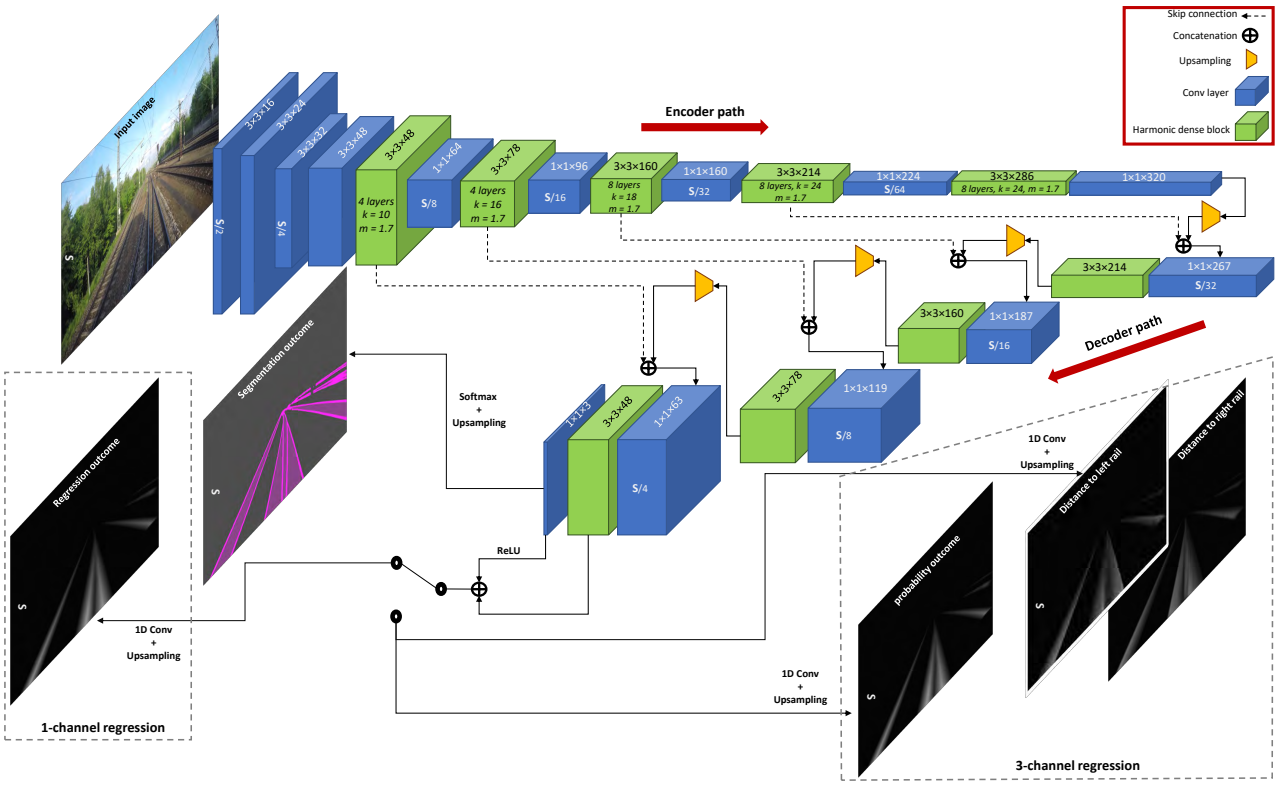

Fig. 6. A detailed structure of the proposed fully convolutional network that outputs regression for triplet coordinates and segmentation within 2D images of rail scene. There are two different designs for the regression branch in our proposed network.

model selection technique and carry out ablation studies, we initially rely on HarDNet [16] that has achieved an acceptable segmentation accuracy at a considerably lower inference time compared with DenseNet [17] and substantially deeper ResNets [18]. The overall structure of the proposed network is illustrated in Fig. 6. Here, one decoder branch performs semantic segmentation [17], while the other branch was designed as a regression module [14] to extract triplets. Two different designs are associated with the regression branch, i.e., the 3-channel regression branch and the 1-channel regression branch. The two are trained separately using distinct sets of ground truth regression data.

Moving from the bottom decoder layer to the top one, four HarDBlocks are observed with the exact same design as the fourth, third, second, and first encoder HarDBlock, respectively. Here, the output of the final decoder HarDBlock f_{HDB} is fed into a 1-D convolution, resulting in a feature map f_{Conv} having the same depth as the number of segmentation classes. Applying Softmax to f_{Conv} followed by a bilinear interpolation upsampling completes the segmentation branch by outputting the segmentation outcome. The regression branch however, fuses f_{HDB} and ReLU activated f_{Conv} features through channel-wise concatenation to leverage semantic information in predicting potentially more accurate heatmaps. The concatenated feature maps are then passed through a ReLU activated 1-D convolution followed by a bilinear interpolation upsampling to produce each regression heatmap. Here, all the convolution layers have the same structure of Conv + BN + ReLU, except for f_{Conv} which is activated depending on the branch type.

C. Track Segment Generation

In this section, we explain the Track Segment Generation (TSG) block. Here, our goal is to slice the input image I into several nonoverlapping subregions, and then, create clusters of track points falling inside each subregion. Each subregion has the same width W as I, while having a small height h as a hyper parameter. We deal with each subregion separately, starting from the bottom most to the top most one, i.e., close to far direction. Clustering within each subregion is performed row by row based on Euclidean distance. In the first row of each subregion, each track point serves as the starting point of a new track segment (cluster) only if the points keep a specific distance from each other. Otherwise, the nearby points fall into the same segment. Throughout the upcoming rows of the subregion, the track points are associated with their nearest segment generated or augmented through the previous row, if the segment and the point lie within a defined vicinity. Otherwise, the track point generates a new cluster. Through the above steps, we generate track segments $\mathbf{S}$ for all the subregions, where $\mathbf{s}_i = \{\mathbf{s}_1^i, ..., \mathbf{s}_{n_s}^i\}$ in $\mathbf{S}$ denotes track segments in ith subregion. Note that to this end, there is no association between the track segments generated in the neighboring subregions. The outcome of generating track segments from the triplets is illustrated in Fig. 2(b).

D. All Possible Ego-paths Generation

This section explains All Possible Ego-paths Generation (APEG) block in which all the possible ego-path trajectories are constructed using the track segments. Firstly, we define a rooted path tree $\mathcal{G}$ which is supposed to incorporate all the topological information of the possible ego-paths. The path tree consists of nodes $\mathcal{N}$ and edges $\mathcal{E}$. The nodes represent topologically meaningful locations within the image frame, falling into three different types: start node, end node, and switch node. The start node represents the starting point of all the possible ego-path trajectories. Here, we look for only one start node, which is expected to lie within a specific distance to the bottom middle of the image. An end node represents the end of a detected possible ego-path. A switch node represents a junction of two different paths in a switch state where two or more paths are split. The edges $\mathcal{E}$ represent path trajectories linking two adjacent nodes.

Creating nodes and edges to shape the path tree is performed through handling the track segments $\mathbf{S}$. Moving from the bottom most subregion to the top one, the segments in the neighboring subregions are associated based on spatial clustering to generate edges $\mathcal{E}$. In the bottom most subregion, the starting point of the closest segment to the bottom middle of the image is the start node of the path tree. Here, the corresponding segment creates an edge. The segments in the following subregions are either associated with the existing edges created in the previous subregions, or discarded. If two or more track segments are associated with the same edge, a switch node is created at the end of the edge, and each segment generates a new edge. Each new edge together with the edge containing the start node will construct a path trajectory. When there is no more track segment to augment an edge, an end node takes place at the end of the edge, finishing off a path trajectory. Finally, having constructed $\mathcal{G}$, all the possible ego-paths are obtained by simply traversing from the start node to the end nodes. The outcome of constructing path tree using track segments is illustrated in Fig. 2(c).

On top of exploiting a semantic segmentation network to regress triplet points locations, we use the segmentation output of the network to reduce regression errors. If a regressed left or right rail pixel does not lie within a specific horizontal distance to a rail class pixel, we shift the estimated location to its nearest rail class pixel with the highest probability in each image row. The specified distance differs in each image row considering the perspective effect. Finally, a polynomial is fitted to each group of left or right rail pixels of a proposed path through least squares curve fitting to obtain a solid visualization of the extracted rails.

E. Loss Function

The dataset chosen to train the TPE-Net and evaluate our rail path extraction algorithm is RailSem19 [19], a segmented dataset with 8500 images covering categories such as rail track and rail area. Also, coordinates of all left and right rail pixels for each track instance are provided. Therefore, using (1) to (4), we reformulate ground truth data of track coordinates to meet our regression task requirements.

We define the multi-task loss as a weighted sum of bootstrapped cross entropy (BCE) [20] loss for segmentation and L1 loss for regressions. BCE loss ($Loss_{seg}$) through each image of the batch is defined as (5). Here, W and H are the image width and height, respectively, ind stands for indicator function, $CE(x, y)$ is each pixel's cross-entropy loss, and t_K is the highest possible threshold such that over all the $W \times H$ pixels, at least K pixels have the indicator output of 1.

$$Loss_{seg} = \frac{1}{K} \sum_{x=1}^{H} \sum_{y=1}^{W} ind(CE(x, y) > t_K) \times CE(x, y) \quad (5)$$

Likewise, regression loss for a heatmap of the batch ($Loss_{reg}$) is obtained through (6). Here, I_{est} is a regression output of TPE-Net, and I_{GT} is the ground truth heatmap. Finally, total loss through each batch is defined using (7) in the case of 1-channel regression, and (8) in the case of 3-channel regression. Here, BS stands for the specified batch size. Also, weight coefficients α, α_d, α_p are introduced through the weighted loss formulas.

$$Loss_{reg} = \frac{1}{W \times H} \sum_{x=1}^{H} \sum_{y=1}^{W} abs(I_{GT}(x, y) - I_{est}(x, y)) \quad (6)$$

$$Loss_{1-channel} = \frac{1}{BS} \sum_{i=1}^{BS} \alpha \times Loss^{i}_{reg} + Loss^{i}_{seg} \quad (7)$$

$$\begin{aligned} Loss_{3-channel} = \frac{1}{BS} \sum_{i=1}^{BS} & \alpha_d \times Loss^{i}_{reg(distances)} \\ & + \alpha_p \times Loss^{i}_{reg(probability)} + Loss^{i}_{seg} \end{aligned} \quad (8)$$

III. Experimental Results

A. Training

Regardless of the baseline, we train TPE-Net using the stochastic gradient descent optimizer on 6000 training and 1000 validation images. The images are resized from the original resolution of 1080×1920 to 540×960. The initial learning rate is 0.001 adjusted by a polynomial decay scheduler, momentum parameter is set to 0.9, weight penalty equals 0.0005, batch size is 8, and the number of epochs is 100. In addition, t_K and K are set to 0.3 and 8192 respectively. To promote synchronous convergence of regression and segmentation losses, we set α, α_d, α_p to 0.4, 0.2, and 20, respectively, which eventually leads to the best network performance within simulations. We train and test the network on a computer with NVIDIA GeForce RTX 3090 GPU, Pytorch 1.10.2, CUDA 12.0, and Ubuntu 18.04 LTS.

B. Performance Metrics

We employ the mean Intersection over Union (mIoU) to evaluate the segmentation performance. However, in the path tree construction, only the possible ego-paths are generated. Therefore, in order to quantitatively evaluate our path proposals extraction algorithm, we need to modify the ground truths so that they cover only the possible ego-paths.

TABLE I

PERFORMANCE OF TPE-NET RAIL PATH EXTRACTION ON 1500 RANDOMLY SELECTED IMAGES OF RAILSEM19 DATASET

TPE-Net baseline	*Regression type*	*Segmentation classes*	*If segmented outcome is used? (to reduce regression errors)*	*True-positive-pixel level*		*All-pixel level*		*Path level*		*mIoU*	
				Average precision	*Average recall*	*Average precision*	*Average recall*	*Average precision*	*Average recall*	*Rail track*	*Rail area*
HarDNet [16]	1-channel	3-class	Yes	0.9207	0.8721	**0.8843**	0.8537	**0.9275**	0.9491	0.7007	0.8974
			No	0.9085	0.8597	0.8730	0.8418	0.9262	0.9477	0.7007	0.8974
		19-class	Yes	0.9056	0.8579	0.8686	0.8432	0.9212	0.9510	0.5847	0.7668
			No	0.8958	0.8477	0.8595	0.8321	0.9212	0.9510	0.5847	0.7668
	3-channel	3-class	Yes	0.8947	0.8667	0.7921	0.8591	0.8408	0.9567	0.6984	0.8952
			No	0.8851	0.8577	0.7846	0.8501	0.8401	0.9557	0.6984	0.8952
		19-class	Yes	0.8673	0.8278	0.8201	0.8137	0.8999	0.9519	0.5965	0.7782
			No	0.8448	0.8067	0.7999	0.7930	0.9001	0.9526	0.5965	0.7782
DLink-Net34 [21]	1-channel	3-class	Yes	0.9104	0.8664	0.7665	0.8568	0.8100	0.9622	0.6424	0.7054
			No	0.8969	0.8531	0.7567	0.8437	0.8102	0.9628	0.6424	0.7054
ERF-Net [22]	1-channel	3-class	Yes	**0.9290**	**0.8864**	0.8451	**0.8756**	0.8818	**0.9701**	**0.7307**	**0.9044**
			No	0.9192	0.8766	0.8371	0.8661	0.8818	**0.9701**	**0.7307**	**0.9044**

Accordingly, we firstly merge all the ground truth triplets of an image to break the association between triplets and their corresponding routes. Here, the resulting data would be in the same shape as the outcome of applying NMS to the network's regression branch. Secondly, the merged data is sequentially fed into the TSG and APEG blocks to filter out the undesired routes and construct the desired ground truth path tree.

Inspired by [12], we evaluate our path proposal generation algorithm in three levels: true-positive-rail pixel level, all-rail pixel level, and rail path level. Through the true-positive-rail pixel level, the true positive estimated rail path is compared point-by-point to its corresponding ground truth. Here, false-positive estimated paths and false-negative ground truth ones are discarded. All-rail pixel level evaluation is done similar to true-positive-rail pixel level, except that here, all the detected and ground truth paths are included within point-by-point comparison. Rail path level deals with whether the whole path is successfully detected or not. To measure the aforementioned performance metrics, a criterion needs to be defined to serve as the matching rate between a pair of a detected and a ground truth path.

Here, we employ F1 score to serve as the matching rate. To measure F1, we first pick the coordinates of each ground truth rail pixel. A window centered by the coordinates is then defined, with the dimension of 10 pixels. If the detected path contains a rail pixel within the defined window, the detected pixel serves as a true positive. Otherwise the ground truth pixel is a false negative. When all the pixels of the current ground truth path are investigated, the unmatched detected pixels add up to the false positives, while the unmatched ground truths fall into the false negatives. Therefore, with TPs, FPs, and FNs in hand, the F1 measure is calculated.

To this end, we have explained how to match the detected paths with the ground truth ones, and consequently, come up with the rail path level performance evaluation. True-positive-rail pixel level performance metrics among the matched paths are measured in the same way as the defined matching rate, i.e., defining a window and looking for pixel correspondences. Here, if we also consider the FPs and FNs of the unmatched estimated and ground truth paths, respectively, all-pixel level precision and recall are measured.

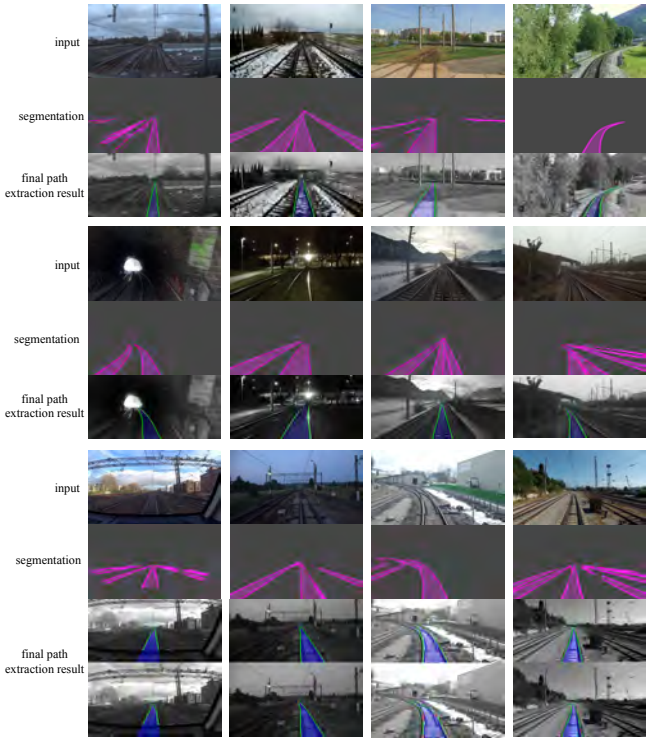

Fig. 7. Visual results of the proposed path extraction algorithm on some sample test images of the RailSem19 using HarDNet baseline and 1-channel regression design. In the final path extraction result, the ground truth rail area is annotated in blue, TP pixels are green, and FP pixels are red. Also, each extracted path is shown in a separate image for clear visualization.

C. Results and Discussion

Numerical results of TPE-Net rail path extraction are provided in Table I. Here, three different segmentation baselines, i.e., HarDNet [16], DLink-Net34 [21], and ERF-Net [22] are trained separately. DLink-Net34 and ERF-

Net were previously employed in the literature to segment the input image into rail tracks and background [10,12]. Each baseline here can be trained using either of the two regression branch versions. Also, through separate trainings, the original 19 segmentation classes of the RailSem19 dataset can be confined to the main 3, i.e., rail area, rail track, and background.

According to the numerical results of the HarDNet baseline, choosing 1-channel regression over the 3-channel design shows a significant boost in robustness and performance metrics due to the higher capability of the network to deal with switch states. Also, comparing the results of 3-class versus 19-class segmentation under the same regression branch design reveals that choosing the main 3 segmentation classes over the original 19 improves segmentation mIoU. Moreover, exploiting the segmentation outcome to make up for the regression inaccuracies contributes to higher average precision and recall in most levels. Therefore, DLink-Net34 and ERF-Net baselines were trained using only 3-class segmentation and 1-channel regression branches as reflected in Table I. Here, the HarDNet baseline inference time is 0.0156, whereas the CPU time of TSG and APEG blocks together on AMD Ryzen 9 5900X 12-core processor is 0.0727 seconds per frame.

The numerical results also indicate that ERF-Net performance stands first in both rail path extraction and semantic segmentation. Also, DLink-Net34 and ERF-Net inference time is 0.0112 and 0.0151 seconds per frame, respectively, making them slightly faster than HarDNet. Here, the closeness of numerical results using different segmentation baselines proves our claim that any segmentation architecture is capable of successfully producing the desired TPE-Net outcomes. Fig. 7 illustrates some path extraction results on RailSem19 test images using 1-channel regression, 3-class segmentation, and HarDNet baseline.

Comparing the proposed rail path extraction algorithm with previous methods is a challenging task due to some gaps in the literature such as dealing with only single-track rail scenes [9], giving rail elements as a whole without proposing instances [1,11], and using private benchmarks [12]. Yet, our method can handle multi-track scenes within a public benchmark while extracting the topography of the possible ego-path instances. Therefore, we have trained different TPE-Net baselines and evaluated their overall performance to provide a fair comparison for now.

IV. Conclusions

Using any off-the-shelf semantic segmentation baseline, the proposed rail path extraction algorithm can successfully deal with switch states by recognizing only the possible ego-paths. In contrast to the previous studies, our network's regression branch can propose left-right rail pixel associations without dependency on the prior knowledge of track configurations, hard-coded geometry-based values, camera parameters, or 3D point cloud. Thus, as a result of understanding the topography of the possible ego-path instances, the autonomous trains can reliably estimate the risk of collision with probable obstacles and make a timely reaction to avoid significant hazards.

References

[1] H. Li, Q. Zhang, D. Zhao, and Y. Chen, "RailNet: An Information Aggregation Network for Rail Track Segmentation," *IEEE IJCNN*, 2020.

[2] S. Belyaev, I. Popov, V. Shubnikov, P. Popov, E. Boltenkova, and D. Savchuk, "Railroad Semantic Segmentation on High-resolution Images," *IEEE ICIT*, v. 60, pp. 2231-2254, 2020.

[3] Y. Wang, L. Wang, Y. H. Hu, and J. Qiu, "RailNet: A Segmentation Network for Railroad Detection," *IEEE Access*, v. 7, pp. 143772–143779, 2019.

[4] K. Kranthi Kumar, M. Dileep Kumar, C. Samsonu, and K. Vamshi Krishna, "Role of Convolutional Neural Networks for any Real Time Image Classification, Recognition and Analysis," *Materials Today: Proceedings*, 2021.

[5] D. He, Y. Qiu, J, Miao, Z, Zou, K. Li, C, Ren, and G. Shen, "Improved Mask R-CNN for Obstacle Detection of Rail Transit," *Measurement*, v. 190, p. 110728, 2022.

[6] Z. Chen, W. Niu, C. Wu, L. Zhang, and Y. Wang, "Near Real-time Situation Awareness and Anomaly Detection for Complex Railway Environment," *Proceedings of IEEE CogSIMA*, pp. 1–8, 2021.

[7] D. He, K. Li, Y. Chen, J. Miao, X. Li, S. Shan, and R. Ren, "Obstacle Detection in Dangerous Railway Track Areas by a Convolutional Neural Network," *Measurement Science and Technology*, v. 32, no. 10, 2021.

[8] T. Ye, X. Zhang, Y. Zhang, and J. Liu, "Railway Traffic Object Detection Using Differential Feature Fusion Convolution Neural Network," *IEEE Transactions on ITS*, v. 22, no. 3, pp. 1375–1387, 2021.

[9] Z. Tao, S. Ren, Y. Shi, X. Wang, and W. Wang, "Accurate and Lightweight Railnet for Real-time Rail Line Detection," *Electronics*, v. 10, no. 16, 2021.

[10] A. Dagvasumberel, B. Myagmardulam, B. Myagmar, B. Luvsankhuu, and T. Nakayama, "Railroad Near-Miss Occurrence Detection and Risk Estimation System with Data from Camera Using Deep Learning," *ICISPC*, 2021.

[11] Z. Wang, X. Wu, G. Yu, and M. Li, "Efficient Rail Area Detection using Convolutional Neural Network," *IEEE Access*, v. 6, pp. 77656–77664, 2018.

[12] S. Yang, G. Yu, Z. Wang, B. Zhou, P. Chen, and Q. Zhang, "A Topology Guided Method for Rail-Track Detection," *IEEE Transactions on Vehicular Technology*, v. 71, no. 2, pp. 1426–1438, 2022.

[13] M. Zwemer, D. van de Wouw, E. G. Jaspers, and P. de with, "A Vision-based Approach for Tramway Rail Extraction," *Proceedings of SPIE*, 2015.

[14] X. Zhou, D. Wang, and P. Krahenbuhl, "Objects as Points," *arXiv preprint arXiv:1904.07850*, 2019.

[15] B. Bischke, P. Helber, J. Folz, D. Borth, and A. Dengel, "Multi-Task Learning for Segmentation of Building Footprints with Deep Neural Networks," *IEEE ICIP*, 2019.

[16] P. Chao, C. Kao, Y. Ruan, C. Huang, and Y. Lin, "HarDNet: A Low Memory Traffic Network," *IEEE ICCV*, 2019.

[17] S. J. M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation," *IEEE CVPRW*, 2017.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[19] O. Zendel, M. Murschitz, M. Zeilinger, D. Steininger, S. Abbasi and C. Beleznai, "RailSem19: A Dataset for Semantic Rail Scene Understanding," *CVPRW*, 2019.

[20] S. Gaj, D. Daniel Ontaneda, and K. Nakamura, "Automatic Segmentation of Gadoliniumenhancing Lesions in Multiple Sclerosis using Deep Learning from Clinical MRI," *PLoS ONE*, 2021.

[21] L. Zhou, C. Zhang, and M. Wu, "D-LinkNet: LinkNet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction," *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 192–196, 2018.

[22] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, v. 19, no. 1, pp. 263–272, 2018.