Monocular Road Damage Size Estimation using Publicly Available Datasets and Dashcam Imagery

Adithya Badidey[1], Ryan Dalby[1], Zhongyi Jiang[1], David Sacharny[2] and Thomas C. Henderson[1]

***Abstract*— Among the challenges of maintaining a safe and efficient transportation system, Departments of Transportation (DOT) must assess the quality of hundreds-of-thousands of miles of roadway every year and prioritize limited resources to address issues that affect safety and reliability. In particular, road damage in the form of 3D analysis of cracks and potholes is difficult to catalog and require significant human resources to survey. However, a new and growing remote-sensing network comprised of low-cost consumer dashcams presents an opportunity to dramatically lower the cost and effort required to perform road damage assessments. This paper provides methods to approach this problem and details a number of public datasets and models that can be used to tackle it. The central contribution here is a set of several practical software pipelines designed to accomplish this task in an automated fashion. An emphasis on deep learning methods is presented that enables organizations to improve or tailor the results according to their specific requirements and the availability of labeled data. Suggestions for possible directions for future work and improvements at each stage of the pipeline are also presented.**

I. INTRODUCTION

Road damage in the form of visible cracking and potholes can pose safety risks for motorists, cyclists, and pedestrians, as well as indicate important information about the state and quality of the road network. Informing transportation departments about where these cracks have formed can enable maintenance crews to effectively prioritize activities and address them before they become a major concern. Currently, transportation departments use a mixture of human-led inspection activities (i.e., driving all the roads) that may or may not include high-resolution data-collection using LIDAR systems. These manual survey activities typically occur on an annual basis, or longer due to the excessive cost and time requirements. Alternatively, a sensing system that is comprised of millions of motorists that are already driving the roads and passively collecting street-level imagery daily could dramatically reduce the cost of maintaining the road network. This new and growing data source, coupled with automated methods presented here represent how transportation authorities can monitor road conditions and prioritize public funds for maintenance. A review of the state-of-the art of automated road damage classification and segmentation, as well as publicly available datasets and neural net models is given here, and we highlight the challenges to the industrial application of automated road damage assessment using consumer dashcams.

[1]University of Utah, Salt Lake City, UT tch@cs.utah.edu

[2]Chief Technology Officer, Blyncsy, Inc., 175 W 200 S STE 1000, Salt Lake City, UT david.sacharny@blyncsy.com

Several problems underlie the application of automated road damage assessment using consumer dashcam imagery, and in particular, the detection of road damage, location estimation, and dimension estimation. Hundreds-of-thousands of vehicles driving the roads in the United States include dash cameras (dashcams) that have the ability to store sample imagery offsite and in datacenters, enabling further post processing of the imagery. The imagery includes different times of day, different intrinsic camera parameters, and usually only a subset of extrinsic parameters (with unknown error), such as GPS location and heading. The challenge is to first detect road damage in the image, to classify it, then to localize it in world coordinates and estimate its size. All steps should be accomplished with minimal human interaction, for example for tuning parameters. A detailed method to approach this problem is presented here, including the datasets, models, and deep learning tasks involved. A description is given of software pipelines for detecting multiple cracks and estimating their size, with experiments that include an implementation. Possible future directions for research are also presented and possible improvements are suggested for each stage of the pipeline.

For a recent review of image-based crack detection methods, see [1]. A broad set of applications and respective methods is presented. However, it must be noted that all of this previous work has as its goal the detection of cracks in images, and not the 3D assessment of those cracks. For example, [2] aims to determine if a pixel is in a crack region in the image; this task is performed quite well on cell phone camera images. [3] attempts to ascertain the presence of a crack in an image; however, there is no ground truth for their dataset except for a small set of manually labeled images. An attempt to overcome noise is given by [4] where a pyramid technique is used for this purpose. [5] provides a method for autonomous vehicles to assess the severity of damage for safety purposes. A semi-supervised learning method is used which combines a class-conditional Variational Autoencoder and a Wasserstein Generative Adversarial Network. Finally, [6] describe a two-camera system mounted in the car which uses a Kalman filter to track cracks in front of the vehicle. As stated previously, our goal is to obtain 3D crack measurements from a 2D image.

II. DATA EXPLORATION

Data for experimentation was provided by a local company (Blyncsy, Inc.) that develops software for transportation authorities. These images vary greatly in terms of camera parameters and camera pose. Some examples of these images

can be seen in Figure 1. Ground truth is generally not available for road damage, and this represents a major hurdle for developing automated crack size estimation. For this reason, publicly available datasets are explored to see if they can be used to train a model that could generalize to a specific dataset.

Figure 2 contains examples of the field-of-view of the different datasets we investigated. Each contains street-level imagery, and they vary greatly in terms of camera perspective. This lack of standard camera parameters further complicates the task since generalizable methods mapping to the target dataset is a difficult, if not ill-posed, task without ground truth.

Fig. 1. Examples of Blyncsy dashcam imagery.

Fig. 2. Comparison of camera FOV of different datasets.

A. RDD2020 Data

The RDD2020 dataset [7] is publicly available and was released as part of the Global Road Damage Detection Challenge 2020, which was a part of the IEEE BigData Cup 2020. The images were collected from Japan, India, and the Czech Republic. The training dataset includes 21041 labeled images, and each image may contain one or more annotations in the form of bounding boxes. Each bounding box represents an instance of road damage and is labeled with one of the following four classes (additional classes in the dataset were deemed irrelevant and discarded).

1) D00: Longitudinal crack
2) D10: Lateral linear crack
3) D20: Alligator crack
4) D40: Potholes

Some training images did not contain any road damage class (no road damage). Some examples from RDD2020 dataset can be seen in Figure 3.

Fig. 3. RDD2020 dataset: Some annotated examples

B. Mandli Data

Mandli UDOT (Utah Department of Transportation) road data consists of monocular road view images of Utah state roads of constant camera parameters and pose along with associated road distress and LIDAR data. This data is collected by Mandli, a private company specializing in mapping road surfaces. UDOT has each state road mapped at least once every year in a single direction, and every few years in both directions in a costly and time consuming process. The mapping occurs using a specialized vehicle that is driven along the roads with a LIDAR camera and a special laser crack measurement system [8]. This data provides a monocular road view imagery but it is important to note the large perspective difference when compared to the input Blyncsy dashcam data as seen in Figure 2.

This data appears very useful as a potential source of ground truth, but there were issues in terms of being able to programmatically access the data outside of the web interface [9]. To overcome this, a Mandli data visualization program was created to directly visualize the data for labeling. Figure 4 shows a custom Mandli data visualization tool developed to explore this data. This program joins the "LCMS" XML file with the associated monocular image and gives visual cues that help in labeling the data. Unfortunately, this data was not explored more because the lack of an easy way to get a large amount of the raw folder data other than a few example folders which were used to create the tool. As a result, no data was manually labeled, although this would be a logical next step if the underlying data were readily accessible. Utilizing this or similar data would be very useful to evaluate the crack size estimation because the LCMS data contains the needed crack size ground truth. This data could also enable segmentation rather than instance detection through manual labeling, which may help in estimating the dimensions of each crack.

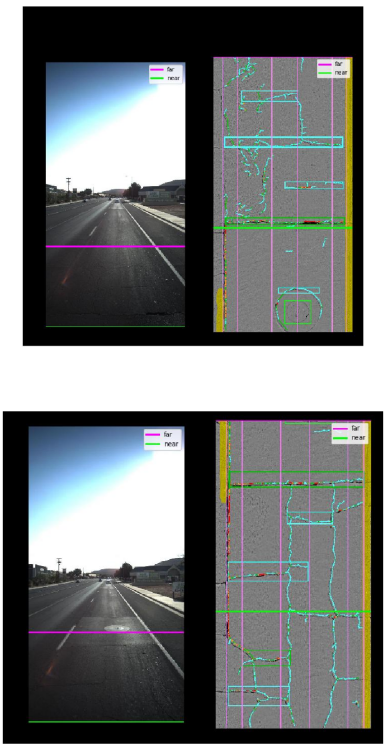

Fig. 4. Mandli visualization tool with two sequential images to illustrate how the monocular image corresponds to LCMS data. Note the manhole cover for reference.

C. Brazilian NDTI Dataset

This dataset was created using data from the Brazilian National Department of Transport Infrastructure (NDTI) [10]. These images were captured using a Highway Diagnostic Vehicle with a fixed front facing 5MP camera. The camera is installed on the highest part of the vehicle, facing the front and with an inclination closer to orthogonality. The visibility of the pavement is 15 meters. The data was developed using these images after filtering out images which present a clear view of the pavement and have some crack(s) or pothole(s). Each image in the dataset comes with 3 masks for each type of annotation: road, crack and pothole (as seen in Figure 5).

This dataset provides a template for a good data collection method for use by deep learning methods. Since the camera is fixed, it enables easier extraction of meaningful information. This dataset was used to perform semantic segmentation using FCN (in subsection III-C) and produced great results.

D. KITTI Dataset

Beyond crack data it was necessary to look at datasets which contain depth information; one of the most commonly used street level depth datasets is the KITTI dataset [11]. The KITTI dataset contains raw sparse LIDAR data, GPS/IMU data, and the associated color and gray-scale stereo image pairs. This dataset can be used for a range of tasks such as depth estimation, depth completion, odometry estimation,

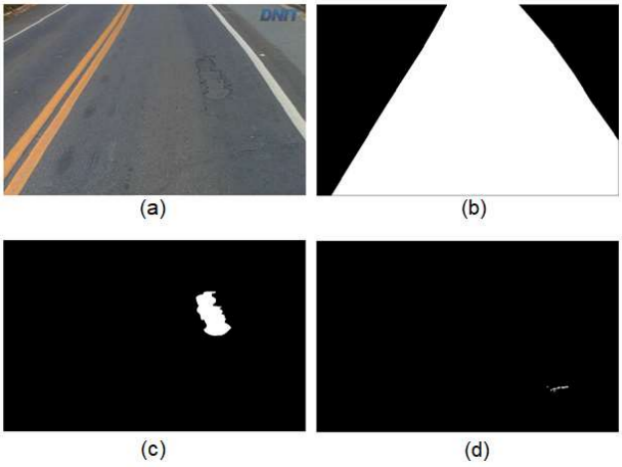

Fig. 5. (a) Example of the original image; (b) the masks corresponding to the road region; (c) pothole; (d) and crack.

optical flow estimation, object detection and more. In relation to this project this dataset was leveraged for monocular depth estimation.

III. Methods Exploration

To determine the 3D crack measurements, various deep learning and computer vision techniques were explored. The lack of ground truth meant we had to look at individual tasks separately (each of them with their individual training data) and then join them to create a pipeline which could solve the complete task. This decision also allowed us to interpret and evaluate the results of each task separately. First we looked at methods to find cracks within images: instance segmentation and semantic segmentation. Instance segmentation identifies each instance of a class within an image and outputs their predicted bounding box and label. Semantic segmentation tries to identify the label of each pixel in an image. Then, we estimate the size of detected cracks using various techniques such as inverse perspective transformation, monocular depth estimation, lane detection, and depth completion.

A. Instance Detection: FasterRCNN

FasterRCNN is an object detection model proposed by [12]. FasterRCNN comes from a family of machine learning models which began with RCNN. These models work in two stages (i) region proposal; and (ii) region classifier. The first of the family, RCNN used selective search to generate region proposals and several SVM classifiers to classify each proposal. The second, Fast RCNN also uses selective search to generate region proposals with CNNs and a method called Region-Of-Interest Pooling (ROIPooling) to extract features for each region proposal for classification. This results in a huge speedup, but the selective search algorithm was still a bottleneck.

FasterRCNN solves bottleneck of region proposal using an algorithm called Region Proposal Network (FasterRCNN has a 10x speedup over FastRCNN). Firstly, FasterRCNN uses a pre-trained CNN backbone as its first step. In case of

PyTorch, the FasterRCNN comes pre-trained with ResNet-50 and ResNet-101 backbones. The Region Proposal Network (RPN) takes the output of the backbone as input and uses a convolution to generate region proposals. These proposals are pruned using non-maxima suppression and passed on to the ROIPooling network which functions as it did in FastRCNN. There are two levels of training involved here: training the RPN to propose regions covering objects rather than ground, and training the ROIPooling to recognize the objects properly. Implementation details are given below.

B. Instance Detection: YOLOv5

YOLO is a family of object detection algorithms that work by dividing images into a grid system. YOLO stands for "You Only Look Once," denoting that it's a single stage detection system (unlike FasterRCNN which is a two stage system). Currently, YOLO has five generations. The latest one is YOLOv5, which is the version we used. YOLOv5 has five different variants: YOLOv5n YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. The only difference is the number of parameters and the complexity of the model. YOLOv5n is the smallest network in the YOLOv5 family. There exists a trade-off here. If the network is smaller, training and inference time will be faster. The network can handle a higher frame rate input in real-time object detection and segmentation tasks. However, its accuracy will be lower. YOLOv5x is the largest network in the YOLOv5 family, and it is also the network we used in our project. Since, in this project, our target data is static images, we could use the relatively large model to pursue higher accuracy.

To train the YOLOv5 network, we must define a "YML" file, including all the class names. In our training dataset, each image links to another label file, which provides for all the detection objects (the four corners of the bounding box and its class). Each label file may contain multiple rows; each row has class name and four 2D points of the bounding boxes. YOLOv5x is the basic model and Adam is the optimizer. A YOLOv5x model was trained for 300 epochs, and the model began over-fitting after 100 epochs. It could reach the highest Precision of 0.5768 (0.5-1 score). The corresponding F1 score was 0.63. In our pipeline, we passed road images into the pre-trained YOLOv5 network. The network gives a few bounding boxes which are used for 3D projection.

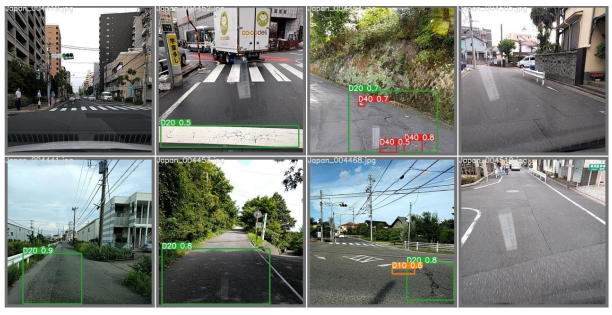

Fig. 6. Detection using YOLOv5 on one example in the test dataset.

C. Semantic Segmentation: Fully Convolutional Networks

Fully Convolutional Networks (FCN) were first proposed for use for semantic segmentation by [13]. FCNs solve the problem of semantic segmentation by having the model first down-sample the image and then up-sample it to get an accurate semantic segmentation. First the image of resolution $H \times W$ is convolved to $\frac{H}{2} \times \frac{W}{2}$ and then to $\frac{H}{4} \times \frac{W}{4}$. Then the reverse operation is done using deconvolution and adding skip-connections. Skip connections are made to allow for up-sampling to use a combination of coarse, high layer information and fine, low layer information. The architecture utilized is available from PyTorch called `fcn_resnet50` that utilizes a `resnet50` backbone.

On the Brazilian NDTI Dataset, we were able to train this model to label lane, pothole and crack segments in images with a f1 score of 0.60. An SGD optimizer with a OneCycleLR Learning rate scheduler was used. However, this model did not generalize to the target Blyncsy dashcam imagery - the results were poor when the model was used on target images.

D. Depth Estimation: AdaBins

A key issue is to estimate depth given a singular monocular image to help inform the road damage size estimation. This is a difficult task given a single view image as depth must solely be inferred by image context. Traditional computer vision methods use geometry of multiple images or views of the scene using structure from motion and stereo vision matching as indicated by [14], however these methods are not effective with a single monocular image. Moving to deep learning methods this problem becomes approachable by leveraging LIDAR and RGB-D data from datasets such as KITTI [11], NYU Depth [15], etc.

Supervised methods directly learn how to map from images to a depth map using a loss function that involves the LIDAR or RGB-D data, the architectures used follow many state-of-the-art CNN networks such as encoder-decoder and ResNet backbones. There is also interest in unsupervised and semi-supervised methods but these methods often require multiple sequences of images to enforce geometric constraints and learn the data distribution and suffer from scale-ambiguity as mentioned by [14].

Currently, one of the state-of-the-art supervised monocular depth estimation networks is AdaBins. AdaBins as proposed by [16] is a standard CNN encoder-decoder backbone network with an added "AdaBins Module" for the head of the network that aims to globally bin depths using adaptive bin sizes. This module utilizes a transformer to perform non-local processing of the high-level features output by the encoder-decoder, aiming to look at things in a more global way. Pre-trained AdaBins models that were trained on the KITTI dataset and the NYU Depth dataset are publicly available.

The primary question was how accurate are relative distances predicted by the model, as absolute distances were expected to be inaccurate. Multiple dashcam images were taken of a scene with a known size object and then correspondences

whose difference in depth value were known were marked. It is important to note that it was assumed, as for many parts of this project, that the depth values are relative to the camera plane, not relative to the camera location (this may be an issue for depth values in the corner of the image), often in the literature it seems that this distinction is not made clear, although it can be assumed for supervised methods with a LIDAR data that the depth values are relative to the LIDAR location. Much of the depth estimation literature does not concern itself with the physical setup of the collected data, but this can be important when it comes to practically applying one of these models and shows the necessity of things like transfer learning to bridge the difference in data.

The experimental dashcam images were run through the pre-trained AdaBins models, bilinearly scaled to match the input KITTI and NYU Depth dimensions, and then once again bilinearly scaled to match the original size. It was found that AdaBins could get somewhat close to the actual object size values but inconsistently at best. It was also found that the AdaBins model pre-trained on the KITTI dataset gave more realistic results than the NYU Depth dataset, this is due to the KITTI dataset being road images while the NYU Depth dataset being primarily indoor scenes. The results can be seen in Figure 7, showing that differences between the estimated and ground-truth depth differentials could be anywhere from around 0.1 meters to 10 meters or more if the chosen point was incorrectly identified as the same depth as the background. This indicates the need to overcome the scale-ambiguity and use depth data that is directly associated with the Blyncsy input images.

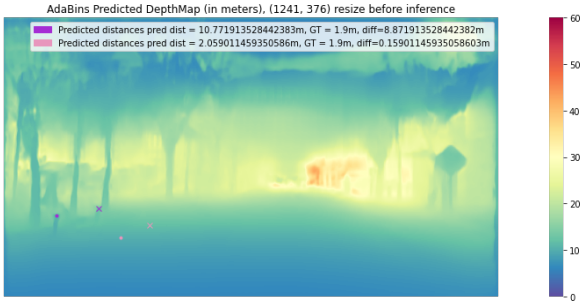

Fig. 7. AdaBins physical evaluation results

E. Lane Detection: PINet

Lane detection using deep learning was another task that was investigated. Lacking ground truth crack information and having a scale-ambiguous problem meant that using a known size object, such as lane width would be a way to describe the width of a crack. A near state-of-the-art network called PINet developed by [17] was used because of its codebase being stable and developed using PyTorch with only Python packages. This network had multiple pre-trained models available that were trained on different datasets; the one utilized for this project was trained on the TuSimple Lane dataset [18]. This network outputs estimated lane points which are classified into being part of a specific lane. It was later found that for fitting a line to these points usage of the Hough transform worked best, and this is discussed in the pipeline section.

IV. Pipelines

Three potential pipelines were exploited to solve this problem.

1) Instance Detection model (FasterRCNN/YOLO) → AdaBins → 3D size estimation (Reprojection/Using lane width).
2) Segmentation → Clustering → 3D size estimation using lane width.
3) Segmentation → Clustering → AdaBins → 3D size estimation using reprojection.

After exploring the required methods in all three pipelines, it was determined that segmentation using FCN did not generalize well to the target dataset and that a better dataset/model was needed. Therefore, we chose to build the pipeline 1 with lane width estimation from pipeline 3 for our proof-of-concept implementation.

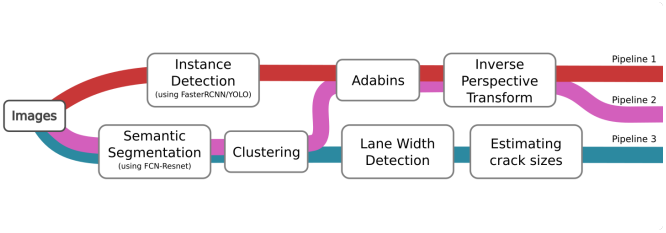

Fig. 8. Pipeline architectures.

V. Implemented Pipeline

A proof-of-concept pipeline was implemented that predicts the bird's-eye view height and width of a crack from Blyncsy dashcam images. The pipeline utilizes crack instance detection through FasterRCNN (YOLOv5 could have also be substituted for it), depth estimation using AdaBins for both crack height estimation and width estimation using the perspective transform, and lane detection using PINet as another estimator of crack width. This pipeline assumes that the bounding boxes tightly encapsulate the crack, and that the crack is parallel to one pair of bounding box edges. If the crack does not meet these assumptions then this method may not be accurate (e.g., a crack that goes diagonally from top right of a bounding box to the bottom left, it would be possible to recover either width or height of the crack from the bounding box dimensions but not both). This also indicates some of the drawbacks of using bounding boxes as opposed to segmentation and also the uncertainty of what crack dimensions really mean (where a crack starts and ends may be subjective). Thus, the goal of the pipeline was formulated as "predicting the physical dimensions of the crack bounding box in world space" (rather than the physical dimension of the crack). A GeoJSON "product" is produced

as the output of this pipeline, and an example output appears in Figure 9.

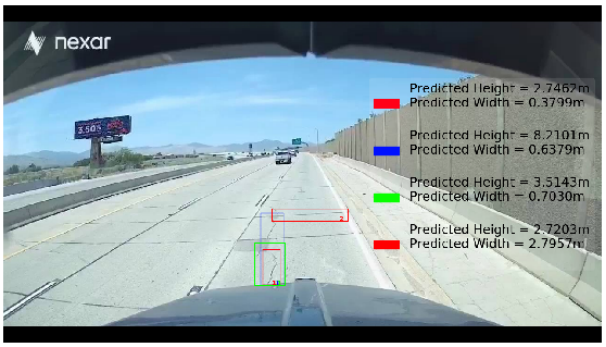

Fig. 9. Pipeline "product" predictions.

A. Crack Detection

To extract the crack's bounding boxes, we trained a FasterRCNN model using a subset of the RDD2000 Dataset (using only the Japan and Czech images). We elected to discard the India images because the road condition were very different from local Utah conditions and image quality was also subpar.

We started with the `fasterrcnn_resnet50_fpn` implementation from PyTorch, which is pre-trained on the Coco dataset. We trained this on our dataset for 5 epochs using a batch size of 32. AdamW was used as the optimizer and OneCycleLR as LRScheduler (with max learning rate of 0.0003). Additionally we added some random cropping, horizontal flip and color jitter to the training images to make the model generalize better. To evaluate the model, for each of the predicted bounding boxes, we checked if it's a true positive or not. Using that we calculated precision and recall for each of the classes and f1-score. The best f1 score obtained was 0.43.

B. Crack Height Estimation Using Depth

To estimate crack height it was assumed that the depth map values were relative to a plane going through the camera. This meant that the difference in depth values along the detected bounding box provides the height of the crack. In implementation, the predicted AdaBins depth was sampled and averaged along the top and bottom edges of the bounding box, giving an averaged top and bottom depth value. The averaged top and bottom depth values were then subtracted from each other to give a prediction of the crack height. The error in this method could not effectively be evaluated but from the physical evaluation of AdaBins this estimate is likely not accurate and needs transfer learning to "scale" the depth map values correctly for the underlying data.

C. Crack Width Estimation Using Inverse Perspective Transform

The method to estimate crack width using the inverse perspective transform assumed fixed camera parameters such as focal length and camera pose. We approximated the "z" homogeneous coordinate for inverse perspective transform using the AdaBins depth map and then conducted the inverse perspective transformation from image coordinates to world coordinates. Thus the units of the AdaBins depth map determined the approximated 3D world space coordinates. This gave the world space coordinates of the bounding box in the same physical units as the depth map. Then the width could be estimated as the difference in the "x" coordinate in 3D world space. In the future, we could use ground-truth depth maps to calibrate the AdaBins model, use known camera parameters specific to each input image, or train a deep learning network to more generally calibrate this transformation.

D. Crack Width Estimation Using Lane Width

The second method to estimate crack width was using lane width. The idea of this method was that lane width could be used as a sort of high level "ruler" in the scene, similar to how humans use objects as visual cues to estimate sizes. PINet was used to detect lanes, it output points representing detected lanes which were clustered into classes using a threshold. It was found that the classification used in the published work often did not perform well enough at correctly separating unique lane lines. Thus, clustering and linear regression techniques were explored. These techniques did not give better results and often had issues overcoming outliers or the tight grouping of points near the horizon.

A more classical line fitting technique called the Hough transform was leveraged which robustly identifies positions of lines (and more generally shapes as described by [19]). This method works by having edge points vote for lines through them in an accumulator discretizing polar coordinate parameter space, then using accumulator maxima to determine lines in the original space. This technique was able to be tuned to robustly fit lane lines to lane points as is illustrated in Figure 10. The Hough transform parameters are sensitive but they were tuned to prevent too many non-existent lines.

From here predicting crack width involves looking at the average bounding box image height value and creating a heuristic which finds the width of a lane in pixels at that average height value. Then the ratio of bounding box width in pixels to the lane width in pixels is multiplied by the lane width's known physical dimensions to give an estimate of the crack width. In the future, transfer learning using Blyncsy dashcam images with labeled lane lines will further improve detection of lane lines. This lane detector could also find many other applications outside of crack size estimation.

E. Discussion

Crack height estimation relies on the accuracy of the depth estimation from AdaBins, which due to scale-ambiguity was not able to be calibrated for the input images. The hope with this approach is that this value can be improved with future transfer learning when depth data for the dashcam images becomes available.

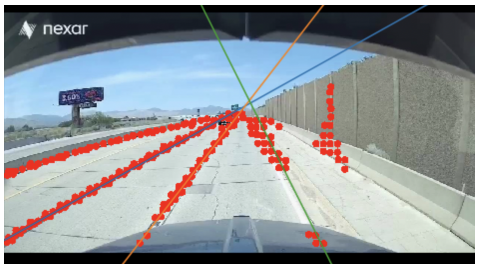

Fig. 10. Utilizing the Hough transform to fit lane lines to lane points.

For crack width estimation it was found that the perspective transformation was much more robust, due to not failing when no lanes were found. Thus to give a single width estimation, if both width estimations were found then they were averaged, or else the perspective transformation was taken. This weighting between these two estimation methods could be tuned in the future when ground truth is available and it is possible to determine which estimate is more accurate. The height estimates were used directly.

Visualizing the results in Figure 11 we can see cases where crack detection and crack size estimation appears to work well in terms of width estimation. In other cases, especially those of crack misdetections the pipeline does not convey much useful information. This also shows how in a multi-stage pipeline errors can compound down the pipeline and why an end-to-end network would be the most flexible.

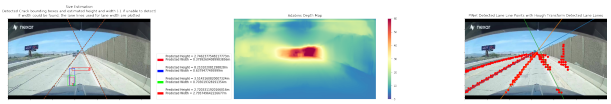

Fig. 11. Full pipeline visualization. From left to right, crack detections with size estimates and lane lines used for size estimation, the predicted depth map, and lane line point detections with Hough transform fit linear lane lines.

F. Pipeline Evaluation

Pipeline evaluation was not directly possible because we had no crack size ground truth so only qualitative evaluation could be conducted. The crack detection algorithm itself is shown to produce good results on the target dataset.

From the results, it can be seen that the crack height estimations are not very physically accurate, for example the length of dashes in dashed lane lines is around 3 meters in the United States [20]. There are many results shown in Figure 11 that appear smaller in height next to dashed lane lines yet are estimated to have heights over 3 meters. Width estimations are much more believable because when both the perspective estimate and lane width estimate are available they are typically within a meter. This indicates that the errors due to the assumed camera parameters of the perspective transform are not too large, but this could not be directly confirmed.

G. Future Work and Improvements

From what we have understood implementing this proof-of-concept pipeline, we can make some suggestions as to how this can be further improved and built into a robust system. First, crack size and depth ground truth data need to be gathered in order to truly understand how well the end-to-end pipeline is performing on the target dataset. There was some progress towards this during this project through using the Mandli data, but issues with having full programmatic access to the underlying data made this route difficult to pursue. The best ground truth data would be associated with images in similar form to the input dashcam images. This data could be collected using a LIDAR scanner, a dashcam, and a way to extract the ground truth from the LIDAR data and annotate the dashcam images with the same. If this ground truth cannot be collected like this, then getting access to the underlying Mandli data or a similar dataset and performing manual annotation would be another route to pursue. Building a robust dataset like this would allow for building and training a network which can do what the pipeline is doing at a single shot.

Second, the current pipeline can be improved in a few ways. For instance detection, continuing training of the model with labeled crack images from the Blyncsy dashcam data would likely result in even better model performance than only using the RDD data. For depth detection, transfer learning must be used to make AdaBins work well with the input dashcam images. This involves collecting associated depth data for the dashcam images. This is also crucial for an accurate inverse perspective transform. To improve the inverse perspective transform, having known camera parameters for each dashcam image the inverse perspective transform would give a better world coordinate estimate. There seem to be some successful efforts at estimating camera parameters directly from the image by [21]. Further investigation in that direction might be fruitful. With enough ground truth it might even be possible to use deep learning to learn an inverse perspective transform that in principle would internally estimate the camera parameters, making it have less error than assuming a single camera model. Lane detection could also be improved using dashcam images with labeled lane lines but using currently available methods, this seems to be less precise than the inverse perspective methods.

Third, the pipeline code needs to be "tensorized" to become computationally scalable. Currently the pipeline will work with a few thousand examples in reasonable amount of time (under an hour) but this is not scalable. Turning most operations of estimating width and height into batch tensor operations would greatly speed up inference as currently these operations a single image at a time. The notebook code also needs to broken into modules and classes to simplify future development.

Lastly, exploring different pipelines that use instance segmentation (as opposed to object detection and semantic segmentation) to identify cracks may be a better way to identify the shape and size of cracks. This would require

development of a new dataset where each crack is segmented individually. This would require implementation of more accurate detection models like MaskRCNN [22] and Detectron2 [23]. The inverse perspective transform too could be leveraged on a per-detected pixel basis.

VI. Conclusion

We present a novel 3D crack size estimation pipeline that uses deep learning to find crack instances and infer depth information from a single dashcam image. Due to the unavailability of ground truth, multiple publicly available datasets and pre-trained models were studied. The UDOT Mandli dataset is particularly promising but there are data accessibility issues to making this a viable option.

Many tasks that would enable this pipeline were investigated including instance detection, semantic segmentation, perspective transformation, depth detection, depth completion, and lane detection. Depth was used as an estimator of crack height and also for crack width when combined with perspective geometry. Crack width was also estimated using detected lanes and known lane widths. This method was not as robust as the inverse perspective transform but has many potential applications outside of this project. Future directions of collecting crack size data, depth data, and segmentation information can improve this pipeline and enable new ways of estimating depth.

VII. Acknowledgements

This project wouldn't have been possible without Blyncsy's partnership and the Deep Learning program's computational and other resources provided by the State of Utah for exploring these deep learning methods. We thank Arad Mohammed, Cathy Liu and Nikola Markovic for their inputs throughout the project. And finally, we would like to thank Michael Butler with UDOT for assistance with the UDOT Mandli data.

References

[1] H. Munawar, A. Hammad, A. Haddad, C. Soares, and S. Waller, "Image-based crack detection methods: A review," *Infrastructure*, vol. 6, no. 115, 2021.

[2] L. Zhang, F. Yang, Y. Zhng, and Y. Zhu, "Road crack detection using deep convolution neural network," in *International Conference on Image Processing*. IEEE, 2016, pp. 3708–3712.

[3] R. Fan, M. Bocus, Y. Zhu, J. Jiao, L. Wang, F. Ma, S. Cheng, and M. Liu, "Road crack detection using deep convolution neural network and adaptive thresholding," in *IEEE Intelligent Vehicle Symposium*. IEEE, 2019, pp. 474–479.

[4] M., H. Zhao, and J. Li, "Road crack detection network under noise based on feature pyramid structure with feature enhancement," *IET Image Processing*, vol. 16, pp. 809–822, 2021.

[5] P. Fassmeyer, B. Funk, F. Kortmann, and P. Drews, "Towards a camera-based road damage assessment and detection for autonomous vehicles: Applying sacled-yolo and cvae-wgan," in *IEEE 94th Vehicular Technology Conference*. IEEE, 2021, pp. 1–7.

[6] Y.-C. Liu, W.-H. Chen, and C. Kuo, "Implementation of pavement defect detection system on edge computing platform," *Applied Sciences*, vol. 11, no. 3725, 2021.

[7] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal, A. Mraz, T. Kashiyama, and Y. Sekimoto, "Transfer learning-based road damage detection for multiple countries," 2020. [Online]. Available: https://arxiv.org/abs/2008.13101

[8] "Laser crack measurement system (lcms)." [Online]. Available: https://www.pavemetrics.com/applications/road-inspection/lcms2-en/

[9] "Roadview explorer 5: Mandli communications." [Online]. Available: https://roadview.udot.utah.gov/utah/index.php

[10] B. T. Passos, M. Cassaniga, A. M. da Rocha Fernandes, K. B. Medeiros, and E. Comunello, "Cracks and potholes in road images," 2020.

[11] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.

[13] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[14] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, "Monocular depth estimation based on deep learning: An overview," *CoRR*, vol. abs/2003.06620, 2020. [Online]. Available: https://arxiv.org/abs/2003.06620

[15] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[16] S. F. Bhat, I. Alhashim, and P. Wonka, "Adabins: Depth estimation using adaptive bins," *CoRR*, vol. abs/2011.14141, 2020. [Online]. Available: https://arxiv.org/abs/2011.14141

[17] Y. Ko, J. Jun, D. Ko, and M. Jeon, "Key points estimation and point instance segmentation approach for lane detection," *CoRR*, vol. abs/2002.06604, 2020. [Online]. Available: https://arxiv.org/abs/2002.06604

[18] TuSimple, "Tusimple/tusimple-benchmark: Download datasets and ground truths: Https://github.com/tusimple/tusimple-benchmark/issues/3." [Online]. Available: https://github.com/TuSimple/tusimple-benchmark

[19] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[20] "Mutcd - 2003 edition revision 1 chapter 3a." [Online]. Available: https://mutcd.fhwa.dot.gov/htm/2003r1/part3/part3a.htm

[21] T. H. Butt and M. Taj, "Camera calibration through camera projection loss," 2021. [Online]. Available: https://arxiv.org/abs/2110.03479

[22] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017. [Online]. Available: https://arxiv.org/abs/1703.06870

[23] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.