Received 21 June 2022, accepted 29 June 2022, date of publication 12 July 2022, date of current version 18 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3190014

APPLIED RESEARCH

Characterization of Semantic Segmentation Models on Mobile Platforms for Self-Navigation in Disaster-Struck Zones

RYAN ZELEK AND HYERAN JEON, (Member, IEEE)
Department of Computer Science and Engineering, University of California Merced, Merced, CA 95343, USA
Corresponding author: Hyeran Jeon (hjeon7@ucmerced.edu)

ABSTRACT The role of unmanned vehicles for searching and localizing the victims in disaster impacted areas such as earthquake-struck zones is getting more important. Self-navigation on an earthquake zone has a unique challenge of detecting irregularly shaped obstacles such as road cracks, debris on the streets, and water puddles. In this paper, we characterize a number of state-of-the-art Fully Convolutional Network (FCN) models on mobile embedded platforms for self-navigation at these sites containing extremely irregular obstacles. We evaluate the models in terms of accuracy, performance, and energy efficiency. We present a few optimizations for our designed vision system. Lastly, we discuss the trade-offs of these models for a couple of mobile platforms that can each perform self-navigation. To enable vehicles to safely navigate earthquake-struck zones, we compile a new annotated image database of various earthquake impacted regions that is different than traditional road damage databases. We train our database with a number of state-of-the-art semantic segmentation models in order to identify obstacles unique to earthquake-struck zones. Based on the statistics and tradeoffs, an optimal FCN model is selected and applied to the mobile vehicular platforms. To our best knowledge, this is the first study that identifies unique challenges and discusses the accuracy, performance, and energy impact of edge-based self-navigation mobile vehicles for earthquake-struck zones. Our proposed database and trained models are publicly available.

INDEX TERMS Convolutional neural networks, edge computing, self-driving, semantic segmentation.

I. INTRODUCTION

With global-wide monitoring and technology evolution, the riskiness of natural disaster is getting lower. However, there are still deadly incidents such as earthquakes. Earthquakes are a geologic inevitability of some countries and states. To reduce the effects from earthquakes, we should think about the ways to quickly search earthquake-struck zones and safely rescue more lives. For this purpose, unmanned autonomous vehicles will be effective, which navigate impacted sites while localizing the people and reporting the damages without risking the lives of others, such as firefighters and other rescue workers. In the natural disaster impacted sites, edge-based small automobiles would be more cost effective and feasible solutions because these small cars can navigate underneath collapsed buildings, unstable bridges, and narrow walkways that are difficult to be driven in by a full-sized vehicle, and also hidden from the viewing angles of aerial solutions such as drones. Though the self-navigation research and industry has been quickly evolving, most of the solutions focus on detecting fairly regular-shaped objects such as humans, buildings, and streets from a full-sized vehicle's perspective. However, these solutions cannot be directly applied to earthquake-struck zones because of the unique obstacles such as debris, cracks, and puddles on the streets, which the mobile edge vehicle perceives in a completely different perspective.

In this study, we present an edge-based unmanned vehicle that can navigate disaster-struck sites by avoiding extremely irregular-shaped obstacles. To understand the design trade-offs, we provide detailed characterizations of recognition accuracy, performance, and energy impact of various

The associate editor coordinating the review of this manuscript and approving it for publication was Oguzhan Urhan.

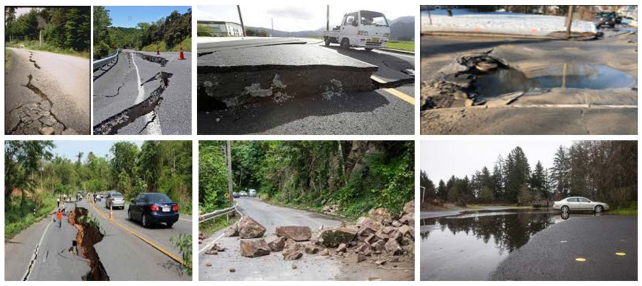
FIGURE 1. A few images from our proposed earthquake-site database.

system components. Convolutional neural network (CNN) is known to be effective for extracting complex features and patterns that can be found in image data [1]. Among various CNN variants, fully convolutional network (FCN) that uses semantic segmentation is known to be effective to detect irregular-shaped objects through pixel-level recognition. Therefore, we develop obstacle avoidance algorithm with FCN. We compare various state-of-the-art FCN models on multiple mobile platforms, which will give insights to the researchers and developers about the pros and cons of different FCN models so that they can choose the right one for their purposes. With these results, we also show a few optimization ideas that speedup the end-to-end execution by 75%. We compile and release a new annotated image database of earthquake-struck sites at *https://gitlab.com/thor-auto1/thor-auto*.

The main contributions of this work are as follows:

- We present a proof-of-concept vehicular system design, namely *Thor* that can navigate disaster-struck sites. It consists of a mobile CPU-GPU System-on-Chip (SoC) platform that can run the full pipeline of self-navigation. The vehicular platform uses a combination of Lidar and FCN recognition outputs to make decisions for collision avoidance at the fields that have extremely irregular-shape obstacles.
- Thorough evaluations and in-depth comparisons are conducted for seven state-of-the-art FCN models in the perspectives of accuracy, performance, and energy-efficiency on two mobile CPU-GPU SoC platforms. To the best of our knowledge, this is the first study that shows in-depth evaluations of FCN models on embedded platforms.
- We present various optimizations to tackle inefficiencies observed from the evaluations of FCN models on embedded platforms. The experimental results show that the optimizations improve the end-to-end performance by 5×.
- We develop a new earthquake-site image database that is annotated with pixel-level labels for semantic segmentation based detection algorithms. Our database contains various obstacles that reside at earthquake-struck sites unlike existing databases that are collections of road cracks for road maintenance purposes.

II. RELATED WORK AND MOTIVATION

A. IMAGE DATABASES FOR SELF-NAVIGATION

In this study, we develop a new image database for navigation in the earthquake-struck site. Though myriad of image recognition algorithms have been developed for self-navigating vehicles, most of them are designed for relatively regular-shaped obstacles such as building, human, dogs, road signs and so on. Cityscapes [12], KITTI [13], and CamVid [14] are the popular databases for self-navigation in city landscapes. They can detect the regular obstacles such as buildings, vehicles, humans, for a full-sized vehicle environment, but they cannot detect unique obstacles seen at the earthquake-struck sites such as road damages, broken bridge, cracks, debris, and unclear street borders.

In an earthquake-struck zone, the most obvious hazard to a vehicle is severely damaged road. There are a number of such databases that can be used to detect road damage and various obstacles on the road, but most of them are designed for road maintenance and hence none can be used for autonomous navigation in earthquake-struck zones, as summarized in Table 1. Road damage detection has been thoroughly studied over the years, but where the end goal was to automate the inspection process and plan the regular maintenance repair [2]–[10], [15]. This application difference is important because it defines the problem and how image features are represented. From earthquakes, the road damage caused by the seismic waves traveling through the earth is much greater than that of normal road degradation, and requires a new methodology for detecting road damage. Most of the current road damage detection solutions aim to detect the smallest of cracks [2]–[4], [8], [9], whereas in our case, such small of cracks would minimally impact the stability of the self-driving vehicle. We consider these minor-sized cracks as generally within millimeters or centimeters wide, whereas the cracks considered for self-navigation in earthquake-struck zones are major-sized and are generally within tens of centimeters wide or larger.

Furthermore, a majority of the current solutions detect cracks from a perspective that is either at or near perpendicular to the ground. Although the clearest of features can be extracted from that perspective, the autonomous ground vehicle sees them completely differently, and its perspective also needs to be considered. For ground-based vehicles, cameras are typically mounted on the front, and face in the forward direction with only a slight downward angle. This type of perspective keeps a safe distance while enabling perception on the road, obstacles, and background.

B. SEMANTIC SEGMENTATION-BASED OBJECT DETECTION

Thanks to the pixel-level object annotation capability, semantic segmentation has been widely used for various object detection applications. A study [16] used FC-DenseNet for landslide detection for the photos taken by satellites in the aftermath of earthquakes. However, their target objects are limited to the landslide regions and the satellite images can

TABLE 1. Comparison of the most relevant databases to ours.

Database	Labeling Method	Camera Perspective	Detectable Classes	Application	Total Images	Crack Severity
GAPv2 [2]	Bounding Box	Perpendicular to ground from top of vehicle height	Road Cracks	Automated road inspection	2468	Primarily Minor
Pavement Distress Detection [3]	Bounding Box	Human height, Aerial view about half a meter high	Road Cracks	Automated road inspection	1200	Primarily Minor
Road Damage Dataset [4]	Bounding Box	Front of full-size vehicle	Road Cracks	Automated road inspection	9053	Primarily Minor
GAPs384 [5]	Pixel-level	Perpendicular to ground from top of vehicle height	Road Cracks	Automated road inspection	384	Primarily Minor
CRACK500 [6] CrackTree [7] CrackForest [8] CrackIT [9]	Pixel-level	Aerial view about half a meter high	Road Cracks	Automated road inspection	500 206 118 56	Primarily Minor
CrackU-net [10]	Pixel-level	Front of full-size vehicle and Aerial view about half a meter high	Road Cracks	Automated road inspection	3000	Primarily Minor
Lost and Found [11]	Pixel-level	Front of full-size vehicle	Small Obstacles on Roads	Self-driving vehicle in regular environments	2104	N/A
Thor Database (ours)	Pixel-level	Front of full-size and miniature-size vehicle, drone height, human height	Road Cracks, Vehicles, Buildings, Water Puddles, Humans, Vegetation, Sky, Other Irregular Objects	Self-driving vehicle in earthquake-impacted environments	737	Primarily Major

not be used for automobile navigation. In [17], they proposed a mapping model which uses semantic segmentation as a component to identify different hazard types and assign a risk to them. For their use case, they also evaluated accuracy of various FCN models including FCN8. However, they did not evaluate end-to-end performance of the tested models and discussed the real-time aspects as their future work, while we provide a more comprehensive evaluations of various models and support real-time navigation capability. The authors in [18] proposed a new model that incorporates an holistically-nested edge detection (HED) network to capture the edges from input images in parallel to a regular FCN. With this additional edge information, their model performs semantic segmentation well on raw images with improved accuracy. However, their model results in a much slower processing time than other FCNs such as FCN8 and SegNet, which have performance issues to be deployed on a mobile device that requires real-time computing. We will show the results later in this paper. The authors in [19] developed a self-driving vehicle to navigate autonomously in urban environments while performing semantic segmentation on four cameras concurrently. They used ERFNet, which is a model that we evaluate in more detail in our paper. Unlike this study that aims at developing a self-navigation model with one specific FCN model, our goal is to understand detailed performance and energy characteristics of various FCN models (including ERFNet) to provide insights to the developers and engineers in selections of a proper model for their goals. DenseNet [20] was proposed to improve computation efficiency of CNNs. DenseNet uses Dense Blocks to realize an interactive concatenation of previous feature maps. Each Dense Block consists of Batch Normalization, ReLU activation, 3×3 convolution filter, and dropout. Instead of the commonly used max pooling technique, they utilize average pooling with stride-net connections to reduce computations on upper layers. DenseNet103 [21], DenseNet10 [22], and DenseNet13 [23] extended DenseNet to support semantic segmentation. The postfix numbers mean the number layers used in each model. DenseNet10 extended DenseNet103 paper when they developed their model for segmenting the different parts of human eyes. DenseNet13 also extended DenseNet103 in a similar way as DenseNet10, but they instead extended it for their application of only segmenting cracks from concrete bridge structures. DenseNet103 extended DenseNet for segmentation especially for urban scene understanding applications where many features and classes are present. Intuitively, DenseNet10 and DenseNet13 are lighter versions of DenseNet103, where each can detect only 2 to 5 object classes. Due to this weaker feature and class detection capability, we evaluate only DenseNet103 in our paper.

There were some applications that used semantic segmentation models for unique environments for robot navigation. A study [24] proposed a new model for performing semantic segmentation on 3D point clouds. Although their model performs with high accuracy on a platform with workstation-level computing power, 3D point cloud data is often very expensive to both collect and compute with, and such data types may not be best suited for low-power mobile platforms, where computing power is extremely limited. The authors in [25] used semantic segmentation as a component to help their robot navigate in greenhouse environments with different types of plants. In their model based on ESPNetv2 [26], the semantic segmentation is used for generic object detection and another algorithm is used for detecting the pixel-level traversability for their navigation. This navigation algorithm is referred to as a Traversability Estimation Module (TEM) and it is also trained by a neural network. Due to a unique environmental requirements, the TEM is trained on data sets that are collected at the same or similar environment where the actual test happens, which is strictly controlled by human operators for the robots. On the other hand, for a disaster impacted area such as an earthquake, the actual test environment cannot be mapped out by the vehicle before the vehicle is deployed for autonomous navigation; objects can be placed anywhere, and traversable paths are rarely organized in straight lines, unlike greenhouses. Their baseline semantic segmentation model, ESPNetv2 does not have superior accuracy than our tested models. For exmample, ESPNetv2 is known to derive higher accuracy than ENet, but lower accuracy than ERFNet [26], while using a comparable number of parameters with ERFNet, but more than ENet. Thus, we do not consider ESPNetv2 for our evaluations. Another semantic segmentation model that was considered was Ghost-UNet [27]. Their model is based on an asymmetry encoder-decoder architecture using Ghost-Net [28] and U-Net [29] and their end use-case was for generic mobile robots. In their work, they also showed it to perform with high accuracy, however, it still contained a considerable amount of parameters more than ENet. Because of that fact and also the fact that no latency or FPS benchmarks were presented in their paper, we couldn't consider this model for our project either, due to practical reasons.

III. EARTHQUAKE SITE IMAGE DATABASE

As there is not a publicly accessible earthquake site image database, we compile a new image database with 737 images that contain objects and scenery that can be found from earthquake-struck sites. The images mainly include classifications of road cracks, water puddles, vehicles, humans, road, sky, vegetation, and other irregular objects such as debris. Of those classes, their categories are broken down into three main types. **Obstacles**, which includes objects that the vehicle should avoid driving into. **Traversable path**, where it is safe for the vehicle to drive through. Lastly, the **undefined** category refers to those classes that do not play a role in obstacle avoidance, which in this work, we consider as the sky and void label types. Void labeled pixels, such as blurry pixels in the far background of some images, are unclear even by human eyes, and such pixels cannot be truthfully labeled with a classification. These void labeled pixels are not predicted by our system and are disregarded from our loss function when training the CNN models. Figure 2 shows the distribution of the classes and categories in our database.

The environment of data collection includes both non-shadowy and partially shadowy environments where direct sunlight may not occur. To achieve a high detection accuracy on edge-based vehicles where the viewing angle from the edge devices is much lower than that of full-size vehicles or drones, we include images from the same angles that can be seen by the edge-based vehicle when it's deployed in the field. We annotate the irregular objects with ground truth labels using PixelAnnotationTool [30]. Our database is used for semantic segmentation and training FCN models.

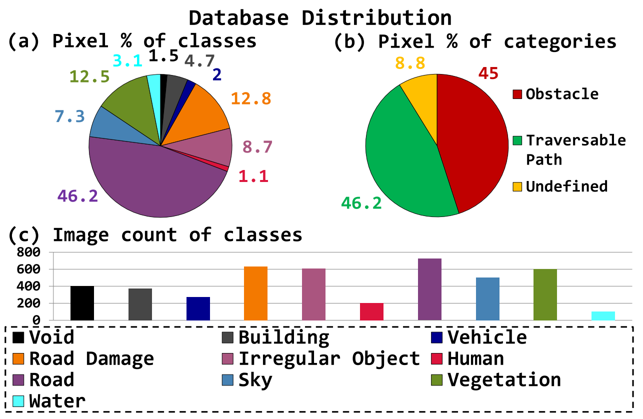

FIGURE 2. Database distribution in terms of: (a) Per-class at a pixel level, in percentage (b) Per-category at a pixel-level, in percentage (c) The number of images where classes are present, out of 737 total images.

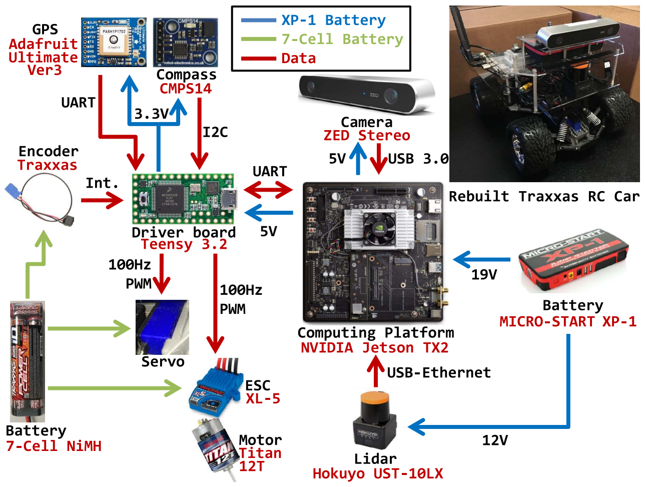

FIGURE 3. Hardware design of the proposed system.

IV. PLATFORM ARCHITECTURE

A. VEHICLE PLATFORM CONFIGURATION

Figure 3 shows the hardware design of the developed edge-based self-navigation vehicle. We test NVIDIA Jetson TX2 [31] and Xavier AGX [32] computing platforms (in MAX-N mode), which are interchangable in our design. The TX2 platform consists of a CPU cluster of a Denver2 dual-core CPU and an ARM Cortex A57 quad-core CPU, while the GPU is Pascal-based with 256 cores. The AGX Xavier consists of an 8-core ARM v8.2 64-bit CPU and a 512-core Volta-based GPU. A sensor fusion of a Hokuyo LiDAR range-finder and a ZED stereo camera is incorporated for the vehicle to perceive the environment at a local level. An Adafruit Global Positioning System (GPS) and an CMPS14 Compass globalize the vehicle. The ZED camera input is fed to the GPU for pixel-level object detection. Robotic Operating System (ROS) Melodic is used as the framework for controlling the various components equipped on the car platform.

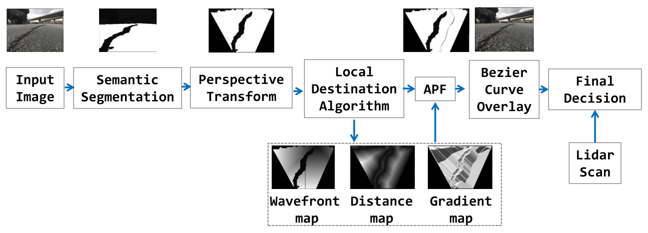

FIGURE 4. **Local navigation software flow.**

B. OBJECT DETECTION AND NAVIGATION PIPELINE

Our vision system relies upon the camera input and Lidar sensor to detect objects at a local level. Fig. 4 shows our software flow for the local navigation task.

Once an image is taken by the camera, the objects are recognized by an FCN via semantic segmentation. Then, the object classes and location information as well as Lidar input are used by the navigation algorithm that finalizes the direction and speed of the driving. The Lidar input is used to ensure the vehicle steers in the safe direction and in a timely manner.

The semantic segmentation CNN task is executed on the GPU while using the Caffe framework. Because this CNN processing has the longest latency and is the bottleneck of our design, we offload the remaining components of our navigation algorithm to be executed on the CPU.

In terms of these CPU-based steps taken after the semantic segmentation, we initially determine an aerial view by applying a perspective transform to the segmented mask, which is then condensed in half to allow real-time operation.

1) LOCAL DESTINATION

From the condensed map, we first determine a safe destination in terms of a pixel-level coordinate. Inspired by the approach used in [33], rows are scanned from the bottom of the mask towards the top. We check all horizontal road intervals that are wide enough for the vehicle to pass through. The center of the largest interval on that row is marked as the local destination. When a row is encountered that has no road interval that's wide enough to fit the vehicle, the row scanning stage is complete.

2) PATH TO THE LOCAL DESTINATION

After a destination is chosen, a safe path needs to be determined that the vehicle can follow. The path we determine is based on an Artificial Potential Field (APF) [34] approach. During APF, the vehicle experiences two potentials, an attractive and repulsive. The attractive pushes the vehicle towards the destination, while the repulsive pushes the vehicle away from obstacles. The attractive force vector (F_{att}) is based on the Euclidean distance from the vehicle (q) to the local destination (q_{goal}), where those two points are defined in terms of pixel-level coordinates:

$$F_{att}(q) = -k_{att} \cdot (q - q_{goal}) \tag{1}$$

To navigate around the irregular shaped obstacles in earthquake-struck zones, we consider the repulsive force vector (F_{rep}) as an accumulation from eight different vectors surrounding the vehicle. These angles (θ_i) consist of all 45° combinations ranging from 0 to 315 degrees. Their magnitude is based on the pixel in that direction's distance to the nearest obstacle (d_{obst_i}), where a distance map is computed by a common Brushfire algorithm. The constants katt and krep were determined through heuristics along with trial and error. If those constant values are not sufficient, then the vehicle will either ignore obstacles or drive around erratically because it would be too sensitive to obstacles. We went through a trial and error process of tuning those constants until the calculated trajectories treated obstacles sufficiently for our application. The term d_0 is directly related to the projected distance away from the vehicle's current location. If there are no obstacles within d_0 from the projected location, then the obstacle's repulsive force does not impact that point of the trajectory. As a result, the repulsive force vector (F_{rep}) can be determined with the following equation:

$$F_{rep}(q) = \sum_{i=1}^{8} k_{rep} \cdot (\frac{1}{d_{obst_i}(q)} - \frac{1}{d_0}) \cdot \frac{\hat{u}(\theta_i)}{d^2_{obst_i}(q)} \tag{2}$$

While the net force vector (F_{net}) is the sum of the attractive and repulsive force vectors:

$$F_{net}(q) = F_{att}(q) + F_{rep}(q) \tag{3}$$

We apply gradient descent to the net force to determine a full trajectory. When obstacles are equally distant from the vehicle or when an obstacle is within a threshold too close, then an alternative repulsive force is applied. This force is based on the gradient map, which provides a one-to-one correlation for the direction in which the nearest obstacle can be found, per pixel. The gradient map is computed by iterating through the distance map and determined based on the minimum value of neighboring pixels. There are a few minor cases where the resulting trajectory cannot complete due to the destination being far away while also being very close to an obstacle, which typically occurs near the upper boundary of the mask. In those cases, we use data from Wavefront algorithm to complete the trajectory. There may still be oscillations due to the nature of APF; to smooth the path, we use four points from this trajectory as control points for a cubic Bezier curve. The resulting curve is smooth enough such that the vehicle can feasibly follow it.

C. SEMANTIC SEGMENTATION MODELS

To understand the impact of different model architectures, we evaluated seven popular FCN models, which are:

- **FCN-VGG16** is proposed by Long, Shelhammer, and Darrell [35] and has shown effective performance for object classification tasks across generic types of applications. Designed with 16 conv. layers and the heavy encoder from VGG16-net, the output is decoded with a pixel stride width of 32. One of the most commonly trained databases for this network is Pascal VOC database [36], which detects animals, furniture, humans, amongst other basic object types from a human's perspective.

- **FCN8** is also proposed by [35] and is an improved version of FCN-VGG16 in that the decoder output stride is taken at eight pixels wide instead of 32. Although this provides additional refinement to the output mask, this is at the cost of requiring additional layers for the decoder.
- **FCN-AlexNet** is again proposed by [35] from their work on converting classical object detection CNNs to FCN networks. Of which, they converted the lightweight eight conv. layer AlexNet model to a fully convolutional model for semantic segmentation.
- **ERFNet** is proposed by Romera *et al.* [37] and was originally designed for the semantic segmentation task for intelligent vehicle types of applications. ERFNet consists of 23 layers but uses dilated convolutions and residual layers as a couple of ways to keep a low memory footprint.
- **ENet** is proposed by Paszke *et al.* [38] and they designed their FCN network specifically for mobile types of applications, including Internet-of-Things (IoT) devices and self-driving cars where computing resources are limited. ENet consists of 29 layers and a few of their efficiencies are gained by the use of decomposed convolutions, such as the dilated and asymmetric convolutions they use in their bottleneck-based architecture.
- **SegNet** is proposed by Badrinarayanan *et al.* [39] and is effective for scene-understanding types of applications, as opposed to generic object detection tasks where background information would not have as much consideration. The 27 conv. layer SegNet model also uses a VGG16-based encoder, however, they improve efficiency by their decoder design, which performs upsampling by reusing max-pooling indices from the encoder.
- **DenseNet103** is proposed by [21] and is also effective for scene-understanding types of applications. The model is designed to have layers to be directly connected together in a feed-forward fashion, through the use of Dense Blocks along with skip connections concatenating down-sampling and up-sampling blocks at various stages of the network. This model was based off the original DenseNet [20] model, and they extended it for semantic segmentation.

We develop all these models in Caffe framework [40] such that each network could be fairly compared to each other by using the same system configuration and software framework. For DenseNet103, we had to implement in PyTorch because it uses extremely large number of layers that are all connected to each other, which makes it unpractical to implement in Caffe's protocol buffer definition file, where each layer has to be manually written. To accelerate model training and improve the accuracy of regular objects such as vehicles, buildings, and humans, we use transfer learning, where each network was initially trained on larger databases (baseline model) and then fine-tuned with our earthquake-site image database.

Hyperparameter selection is determined based on the parameters used by the baseline models and then optimized to achieve the best prediction accuracy with our earthquake image database. Table 2 shows the hyperparameters that derive the best accuracy for each model. We use CUDA 11, cuDNN 7.6, and an NVIDIA TITAN RTX GPU for training. As a loss function for training, we incorporate a SoftMax with loss layer and normalize it across the batch size. As regularization avoids overfitting the models, we train each model until the validation loss no longer decreases and the accuracy becomes stable at an optimum. Images were rotated along the vertical axis to augment the data and further improve generalization.

In summary, before training on our earthquake database, the models were initially trained from scratch on larger databases (listed in the bottom row of Table 2) with a set of hyperparameters best suited for those models. This is what we refer to as *pre-training*. After the pre-training is complete, then we fine-tune the pre-trained models with our earthquake database with the best hyperparameters that we found (also listed in Table 2) from our preliminary experiments in a process that we refer to as *transfer learning*.

Our earthquake database is divided into two parts, a training set and a validation set. The training set consists of 687 image-label pairs while the validation set consists of 50 image-label pairs. All the models we evaluate are trained with the same training set and tested with the same validation set.

V. CHARACTERIZATION AND ANALYSIS

A. ACCURACY

We first evaluate prediction accuracy of the seven FCN models with 50 testing images that encompass all of the object classes in environments with both a high and low number of classes present. Figure 5 shows a few visualized segmentation results of each network for the earthquake-site database. The top row shows the image inputs that include various obstacles such as cracks (the first and the fourth columns), another vehicle (the third column), and debris (the fifth column). The second row shows the ground truth where cracks, other vehicles, and debris are indicated with orange, blue, and light violet colors, respectively. The safe regions for driving are marked with dark violet colored pixels. As can be seen, most of the critical and large obstacles are well recognized by all networks, while some networks show poor pixel-level coverage even for the large obstacles and mis-classification for extremely irregular obstacles such as debris. For example, in the right-most column of Figure 5, a pile of debris is correctly detected and classified by FCN-VGG16 and ENet. The other networks correctly recognized the regions of debris but mis-classified them as cracks. Likely, the crack region on the fourth image is almost completely recognized by FCN8,

TABLE 2. Hyperparameters used when training each model.

	FCN8	FCN-VGG16	FCN-AlexNet	ERFNet	ENet	SegNet	DenseNet103
Iterations	240,450	480,900	240,450	58,182	164,193	126,474	300,219
Epoch	350	700	350	1,016	956	920	874
Learning Rate	1e-4	1e-4	1e-3	5e-4	1e-4	1e-3	1e-3
Batch Size	1	1	1	12	4	5	2
Weight Decay	5e-4	5e-4	5e-4	2e-4	1e-4	5e-4	1e-4
Solver	SGD	SGD	SGD	Adam	Adam	SGD	RMSprop
Momentum(2)	0.9	0.9	0.9	0.9 (0.999)	0.9 (0.999)	0.9	0
Pretrained Database	Pascal VOC [36]			Cityscapes [12]			CamVid [14]

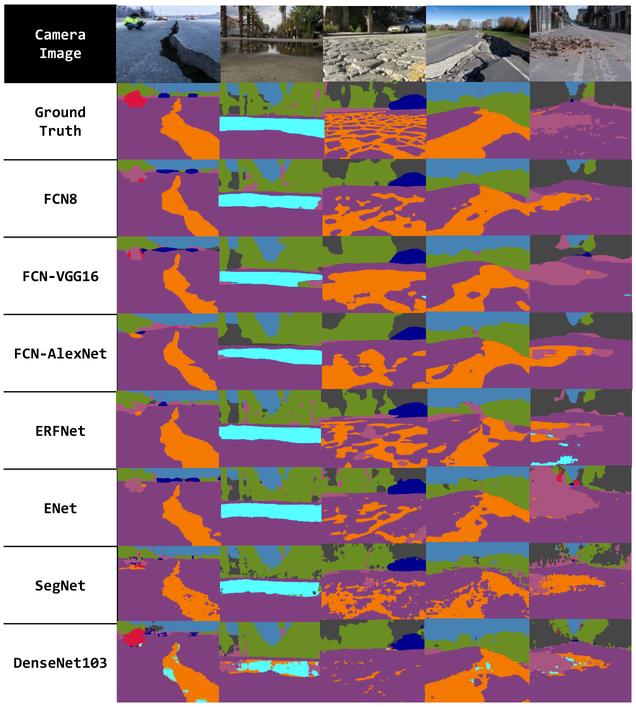

FIGURE 5. Visualization examples of image segmentation used as a method for qualitatively comparing and evaluating the accuracy of the models.

FCN-VGG16, and FCN-AlexNet, while notably higher number of pixels in the crack regions are classified as safe road by ERFNet, ENet, SegNet, and DenseNet103.

However, we observed that the pixel-level accuracy does not necessarily lead to a collision with the obstacle. As far as the coarse-grained region of the obstacle is recognized as not-passable obstacle, the navigation algorithm made a detour to avoid collision. For example, in the example of debris recognition result, even when some networks recognized the debris as crack, as both are not passable obstacles, the vehicle did not pass over them. Similarly, in the example of the crack recognition result, even when some pixels are classified as safe regions, the car did not fall into the crack because majority of the pixels were correctly classified as crack. In image recognition algorithms that use semantic segmentation encounter, so called a *pixel to application accuracy disparity*. In other words, pixel-level recognition accuracy may not be the same with application-level accuracy. To accommodate the accuracy disparity, we measure the accuracy in two levels: pixel-level and object-level.

Deciding upon the optimal accuracy metric for semantic segmentation models is not a completely solved problem and is still a topic being researched in the community. For example in [41], they attempt to solve this problem by developing a single accuracy metric that considers both global classification and contour segmentation when defining their proposed singular metric. For the pixel-level accuracy evaluations in our work, we use the already well-known accuracy metrics such as Intersection-Over-Union (IoU), Global, Precision, Recall, and F1 as equated like below, where TP, FP, TN, and FN are true positive, false positive, true negative, and false negative, respectively. With pixel-level accuracy, we can understand how correctly each pixel is classified. The results are shown in Figure 6 (a) where IoU takes the average of all classes. The analysis of the results will follow shortly.

$$IoU = \frac{TP}{TP + FP + FN} \tag{4}$$

$$Global = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = \frac{2 \cdot Prec \cdot Recall}{Prec + Recall} \tag{8}$$

As noted earlier, pixel-level accuracy does not provide the recognition effectiveness for the navigation. Therefore, we propose to further break down the classes into two categories:

1) the traversable paths and 2) the non-traversable paths that have obstacles that cause collisions. To distinguish this from the original pixel-level recognition, we call the original pixel accuracy as *class-based* and this new category accuracy as *category-based* recognition. With category-based

recognition, we can understand the prediction accuracy that lead to effectiveness of the navigation algorithm.

Figure 6 shows all five metrics in (a) class-based and (b) category-based recognition. Notable differences are observed in mean IoU where category-based recognition show a significantly higher accuracy than individual pixel class-level recognition. This means that navigation decision will be made with 80% accuracy even when 40% of pixels are recognized in incorrect classes. Though most of the metrics show higher accuracy in category-based recognition, the Global metric drops about 10% in category-based recognition. This reduction in global accuracy can be explained by the TN bias associated with the class-based measurement due to its much higher number of classes. Note that TN scores represent pixels where the specific class under test correctly (truly) predicted the pixel as not belong to its class (negative). This means that for images with a large number of classes, the per-class representations would be dominated by the not-belonging (negative) pixels and TN pixels would increase as the number of classes increases. Therefore, lower category-based Global metric is sourced by significantly fewer category-based classes.

In terms of the other metrics, the precision was around 5-10% better than the recall in both recognition methods. This precision vs. recall difference implies that the models are slightly better at predicting positive instances of the objects than they were at capturing the entirety of the ground-truth objects. This property benefits scenarios such as self-driving with irregular shaped objects, where the objects need to be detected, but their exact shape and size is not necessarily needed to still be effective for the navigation.

Among the networks, FCN8 consistently showed outstanding accuracy in both recognition methods, while FCN-AlexNet and SegNet were the worst. The highest accuracy with FCN8 is mainly attributed to the largest number of parameters it computes in addition to the low pixel stride it takes at the output (eight pixels). The number of parameters each model computes is shown in Table 3. FCN-VGG16 and FCN-AlexNet instead take the output at a larger stride of 32 pixels, which contributes to their lower accuracy. The per-layer parameter breakdown is plotted in Figure 10. On the other hand, though ENet and ERFNet stride by only two pixels at the output, they show lower accuracy than larger models because they are originally designed for supporting extremely resource-limited devices with significantly fewer parameters rather than achieving higher accuracy. SegNet also computes a larger number of parameters similar to FCN8, but its decoder scheme is completely different from all the other networks. Unlike the other networks that upsample through transposed convolution layers, SegNet does not use transposed convolution layers for upsampling. Instead, SegNet reuses pooling indices from the encoder's max pooling layers, which led it to a lower detection capability compared to the other models.

In the object-level accuracy, we assume that an object is correctly recognized if the majority of the object region is

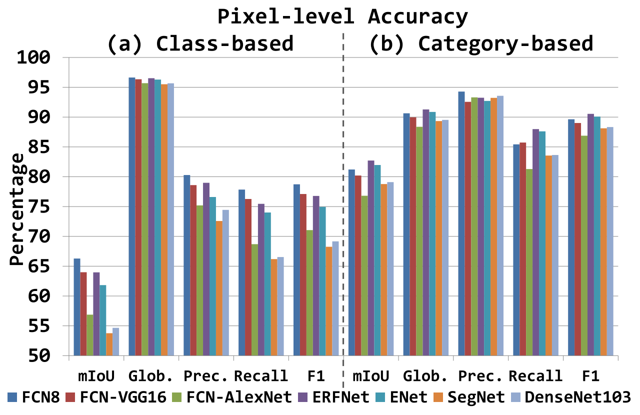

FIGURE 6. Pixel-level accuracy in terms of (a) Class-based taken from the mean of all classes (b) Category-based taken of obstacles.

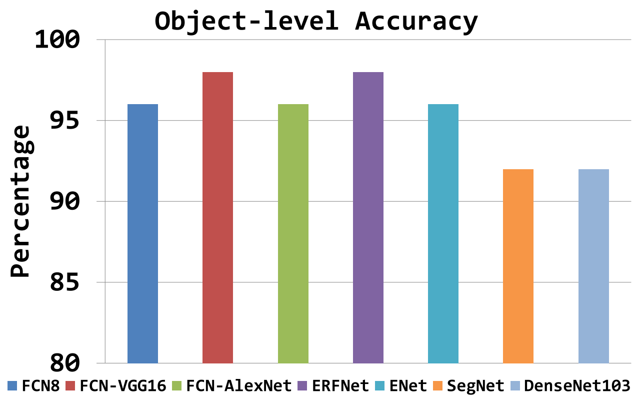

FIGURE 7. Object-level accuracy of obstacle detection.

classified as the target object. In this case, some pixel-level prediction errors are ignored. Although the pixel-level metrics are useful for comparing segmentation models, the exact shape and size is not necessarily needed to effectively detect an obstacle, especially for the disaster-struck sites that have extremely irregular-shaped objects. Thus, we also evaluate the detection capability at an object-level, inspired by [42] and [33], we use 50% coverage as the threshold to determine object-level detection accuracy. These object-level results are shown in Figure 7.

There are a few borderline cases where camera-based vision may fail. In an earthquake-impacted site, the most relevant case would be where road is broken up in such a way that it has an altered height-angle with respect to the vehicle. In those cases, the features from the road cracks may not be visible from the models or even human eyes. To handle such cases, our system incorporates a Lidar sensor which can detect the severely elevated or broken up road amongst the other objects that cannot be fully detected by camera-based vision. By using the combined inputs from an FCN model and a Lidar, our car platform showed zero obstacle collision at the field tests.

Observation 1: *Though most of the tested models show above 70% accuracy in all accuracy evaluation metrics, the* *models using more parameters and shorter output stride showed better accuracy.*

***Observation 2:** Pixel-wise prediction accuracy misleads the navigation accuracy due to the emphasized true negativeness. Category-based prediction accuracy that classifies objects into binary (either traversable or not) can provide more reliable and stable accuracy results for navigation systems.*

B. PERFORMANCE

In this section, we show detailed system-level performance of the tested models on SoC platforms. We compare the seven models in all experiments except for some experiments that have influences of framework such as real-time throughput and power consumption. For those experiments, we exclude DenseNet103 for fair comparisons. Note that DenseNet103 uses PyTorch while the other models are developed with Caffe. As different frameworks can incidentally bias the performance of one model over the other, we believe it is unfair to make a direct comparison among the models that use different frameworks.

To recognize obstacles while driving at real-time, the recognition throughput capability in frames per second (FPS) unit is used as an important performance metric in self-navigation systems. The recognition performance is determined by both the efficiency of FCN and the processing hardware. Therefore, we evaluated FPS of the six FCN models on two edge-based GPU platforms that are NVIDIA TX2 and Xavier AGX. Figure 8 shows the results. As can be seen from the figure, Xavier consistently computes with a higher throughput than TX2, which can primarily be attributed to its double the amount of CUDA cores, which stacks up at 512 CUDA cores when compared to the TX2's 256 CUDA cores. These CUDA cores are important because these cores are directly responsible for the convolutional layers in the CNN models. The breakdown of convolution vs non-convolution execution time is shown in Fig. 9, which shows that the convolutional layers can dominate the model's computing cost and directly relates to the execution time of the models, and stresses the importance of having a large number of CUDA cores in the platform. The global memory is also important, because that is responsible for holding the weights of these models during execution. TX2 operates with a 128-bit 8GB LPDDR4-3732 memory interface while Xavier operates with a 256-bit 32GB LPDDR4x-4266 memory interface that is both faster and more energy efficient.

Interestingly, FCN-AlexNet, ERFNet, and ENet show 2-3× better throughput than the others on both platforms. To understand this notable performance difference, we further investigate the per-layer parameter size and latency breakdown as shown in Figure 10. As can be seen, one of the notable characteristics of the three models is significantly less parameter usage. As the right-hand side Y-axis and the violet-colored line graphs shows, ERFNet and ENet use almost 300% less parameters than FCN8 and FCN-VGG16. FCN-AlexNet uses more parameter than SegNet but runs

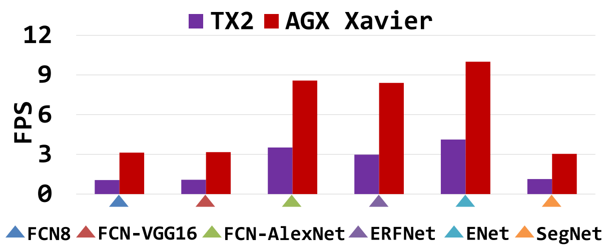

FIGURE 8. Baseline performance comparison of the FCN models in terms of FPS.

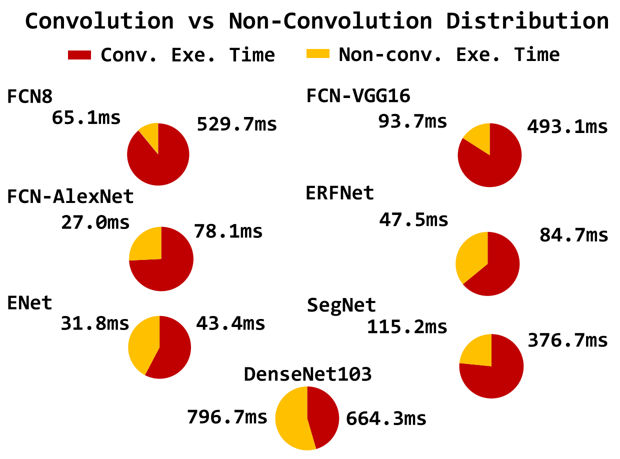

FIGURE 9. Convolution vs non-convolution distribution.

TABLE 3. Total number of parameters computed and size of each model.

	FCN8	FCN-VGG16	FCN-AlexNet	ERFNet	ENet	SegNet	DenseNet103
No. Params. (Mil)	135	137	59	2.1	0.37	30	9.32
Model Size (MB)	514	520	224	7.96	1.47	113	38

almost 30% fewer layers, and hence layer function invocation latencies are effectively eliminated. Note that each layer needs at least one GPU kernel execution, which also require input and output data transfer. Therefore, FCN-AlexNet achieves good performance by reducing kernel invocation overheads.

Next, we breakdown the performance on an end-to-end level, from when a camera frame is received by the ROS node executing the CNN, and until a separate ROS node determines a vehicle direction based on that mask and our navigation algorithm. In pre-processing stage, we convert the camera frame to a 32-bit three-channel RGB float, then reshape and wrap it onto the input layer of the network. The forward-propagation delay includes the GPU processing of the CNN. The post-processing mainly includes pulling from the CNN output layer and formatting the data into a OpenCV grayscale matrix. Our navigation algorithm uses the generated mask by the CNN to determine a path for the vehicle to travel, which operates in parallel to the other ROS nodes. Figure 11 shows this timing breakdown for the fastest network that we tested in ENet.

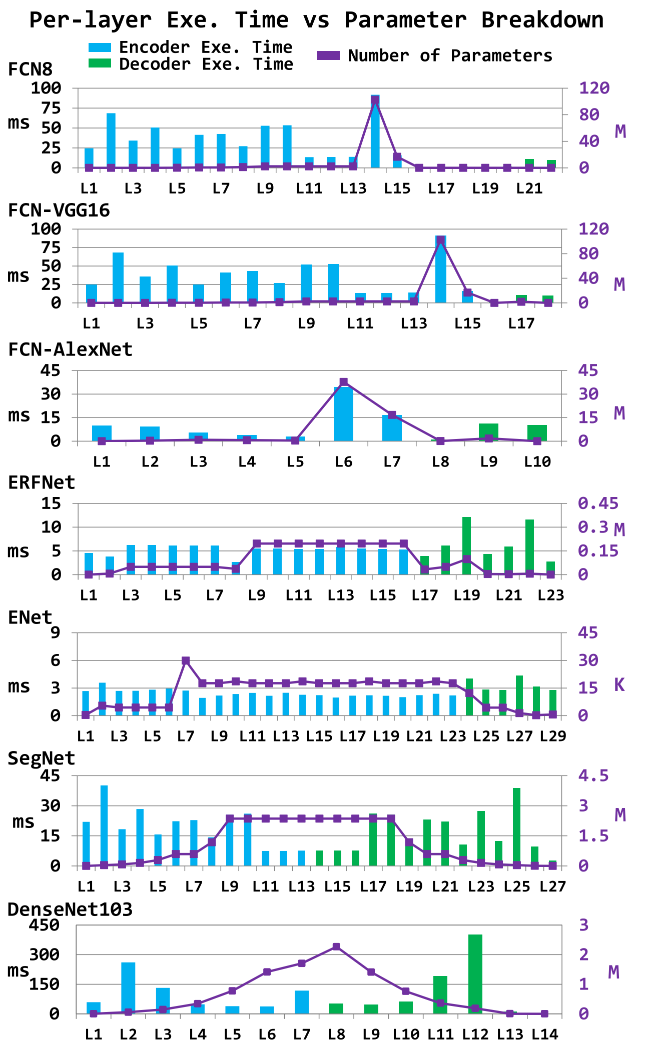

FIGURE 10. Per-layer execution time vs parameter breakdown comparing the networks. In ENet [38] these layers are referred to as modules. In DenseNet103 [21], these layers represent a summary of the various blocks in that model.

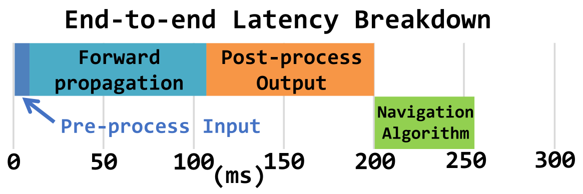

FIGURE 11. End-to-end latency breakdown (with ENet deployed on the TX2).

As CNN is the most compute-intensive function, we expected forward propagation time dominates the overall latency. However, surprisingly, post-processing stage took almost as long of time as the actual forward-propagation. To find the root cause of this long latency, we dug the code of Caffe and NVCaffe and found that the last layer of CNN is not well parallelized in both frameworks. Though the last layer requires a high-level of parallelism because it traverses all the pixels to find the proper label for prediction, the frameworks run them on a CPU. One reason may be to reduce memory copy latency by running the last layer on CPU, which can directly serve the final prediction in CPU side system memory. This optimization may work for server systems that GPU uses a separate device memory. But, in the mobile systems that use a unified shared memory, the CPU-side computation only incur performance overhead. We provide more details about this and propose an optimization in Section VI.

Observation 3: *The number of layers and the amount of parameters have strong impact to the overall throughput. Though more layers and parameters help increase accuracy for complex objects, they need to be trimmed if throughput is the first-priority design consideration, which is true for the systems that need real-time performance.*

Observation 4: *From the end-to-end latency breakdown, the CNN post-processing needs to be carefully handled or else it can significantly extend the end-to-end latency. Many open-source CNN benchmarks and reference codes commonly overlook or ignore this important step, especially because most of the frameworks are designed for server systems that has different CPU-GPU organizations than mobile systems.*

C. ENERGY EFFICIENCY

To understand the energy efficiency of the models with our system, we break down the power consumption of each component, from when the vehicle is in an idle state and until it is steadily active while performing inference. We conduct measurements while reading from six different INA3221 voltage/current sensors on the Jetson, approximately every 300ms. Figure 12 shows this sequence and the associated power dissipation with ENet deployed on the system. The 5V rail powers the ZED camera, USB-Ethernet Adapter for Lidar communication, GPS, Compass, USB hub, and the Teensy uC (in descending order), all of which consume a relatively constant amount of power in our design. CPU handles navigation control along with ROS node management while GPU is dedicated towards CNN processing which operates in parallel to the other ROS nodes. Both Jetson platforms are designed with unified memory, where the LPDDR4 is shared between GPU and CPU. The SOC is responsible for a few background operations specific to the Jetson design, such as managing the BPMP (Boot and Power Management Processor) and IRAM.

Because CNN is always executing on GPU, the CPU-based navigation control is limited by the throughput of the CNN. This GPU-to-CPU dependency raises the question as to how the CNN speed affects the power consumption of GPU and CPU at the system-level. In Figure 13, we breakdown the power of each component and CNN model, as well as the accumulated system-level power. Different colors indicate different models. As can be seen in Figure 13(a), CPU and GPU dominates power consumption, though there are some differences among the models especially in GPU power. However, one notable finding from this per-component power consumption is that a Lidar consumes as much power as the two processors and even more than CPU.

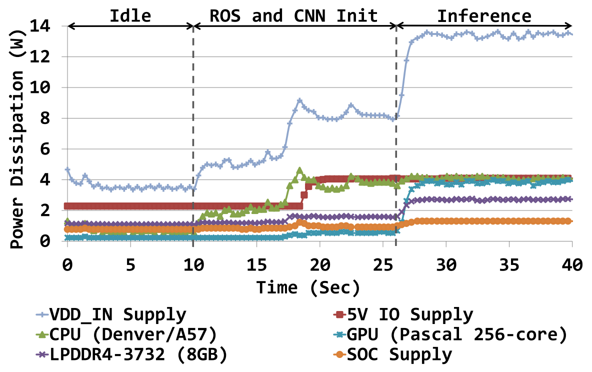

FIGURE 12. Power monitoring sequence with ENet deployed on the TX2.

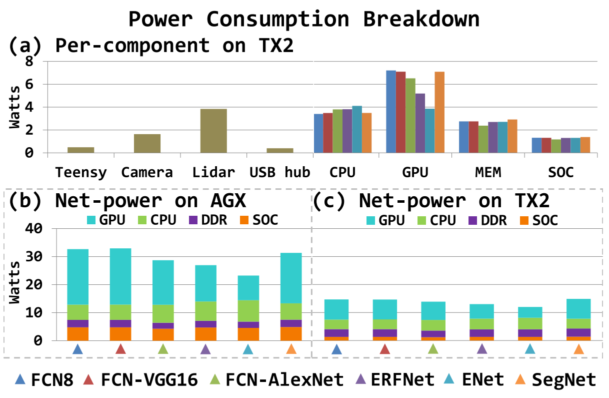

FIGURE 13. System and CNN power dissipation (a) Components on the TX2 (b) AGX Xavier accumulation (c) TX2 accumulation.

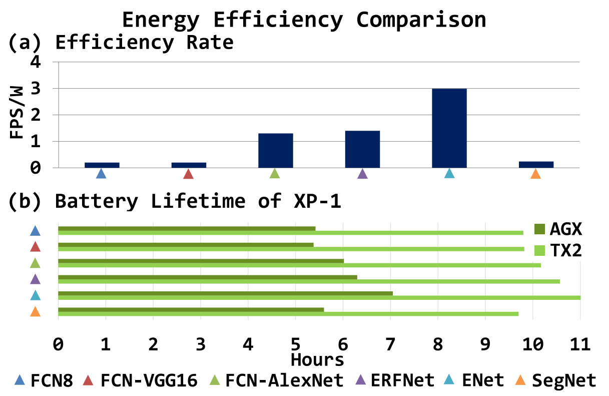

FIGURE 14. Energy efficiency comparison of the CNN models in terms of: (a) Frames per second per watt (b) Maximum battery lifetime of the XP-1.

Considering the net power consumption with the models, power was generally lower as the model speed increased. We found an exception to this trend when comparing FCN-AlexNet and ERFNet. FCN-AlexNet was slightly faster, however, it consumed about 1.4W more from GPU than ERFNet. We then dug deeper to try and understand the efficiency rates of the models, which we measure and report in Figure 14, along with the maximum battery lifetime of our system comparing each model and platform.

Observations 5: *From the power consumption measurements with ENet, the Lidar sensor (including all of its components) consumes a very comparable amount of power compared to the GPU chip alone.*

Observations 6: *From the energy efficiency comparisons, we find that model speed is a good indicator of its energy efficiency, but not necessarily a direct correlation.*

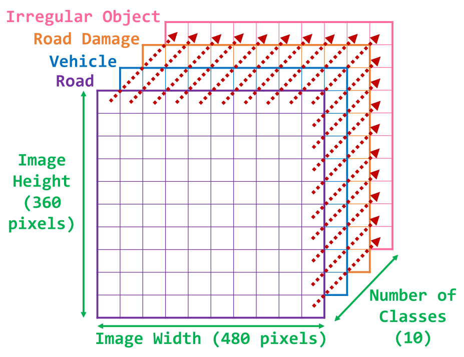

FIGURE 15. Argmax visualization example, diagram inspired by [44].

VI. OPTIMIZATIONS

A. ACCELERATION OF ARGMAX LAYER ON GPU

We found an opportunity for Caffe [40] and NVCaffe [43] to be optimized, which was also discovered by [44] in 2018, but unaddressed by either Caffe developer, up until at least this paper. From the statistics of the design mentioned in the Performance section, we found a significant bottleneck while the network output was being post-processed, which turned out to be when calculating the argmax of the CNN's final layer. During the argmax stage, all of the possibly predicted classes are iterated through for each pixel, and the index consisting of the largest value contains the final prediction for that pixel. Figure 15 shows a visualized example as to how the iteration for this processing is done on a 2D image with multiple classes, where the red arrows represent argmax computations being done on each pixel of the image, and over each class. From digging through the source code of Caffe, we found that this layer was being processed by the CPU, which computes the argmax primarily sequential for each pixel. Given that our network output mask was 480 $\times$ 360 pixels, this adds up to over 1.7M sequential operations. Since the argmax operation for semantic segmentation does not have dependencies among neighboring pixels, each pixel does not need to be checked sequentially. By fully parallelizing the pixel-wise argmax towards the GPU, we can allocate one CUDA thread per pixel to achieve a significantly higher performance improvement. Figure 16 shows the end-to-end timing breakdown of our system comparing both CPU-based and GPU-based argmax implementations.

Additional performance results of the various models comparing the CPU-based argmax with the GPU-based argmax are shown in Fig. 17a, when tested on our Jetson platforms with CUDA 10 and cuDNN 7.5. Fig. 17b shows the resulting end-to-end latency on our Jetson boards with our optimized GPU-based argmax implementation. Our mask-based local navigation algorithm accounts for about 56ms of the

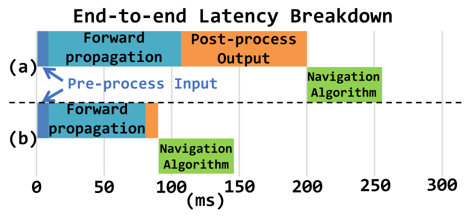

FIGURE 16. Optimized end-to-end latency breakdown with ENet deployed on the TX2. (a) Original latency using CPU-based argmax (b) Optimized latency using GPU-based argmax.

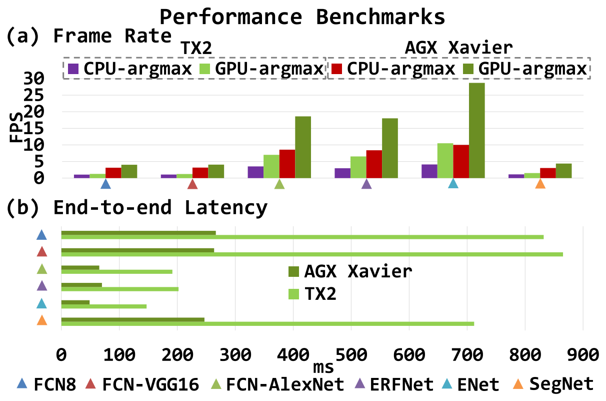

FIGURE 17. Optimized performance with the CNN models (a) FPS (b) Optimized end-to-end latency.

reported TX2 latencies and about 22ms of the AGX Xavier latencies.

We also experimented the models with TensorRT 7.1 (latest compatible with JetPack, at the time of this writing) and found that they didn't support the argmax layer. However, from experimenting with their reference code, the results indicated that they computed the argmax via CPU while using additional host-side code.

B. BOOSTING FPS WITH SENSOR FUSION

Sensor fusion is an emerging approach in the self-driving community to further help improve the reliability and capabilities of vision systems onboard autonomous vehicles. For example, the authors in [45], [46], and [47] showed an enhanced perceptive system accuracy when both Lidar and Camera sensor data were fused together.

To continue accelerating the system scan rate and ensure rapid updates to our vehicle, we apply a sensor fusion by using the segmented mask from our camera and our Lidar sensor, as shown in Figure 18. Yellow cells represent data from CNN and Lidar sensor. Light green cells represents checks in our algorithm that look at the magnitude and direction of the CNN-based trajectory while light blue cells represent checks that look for obstacles detected by the Lidar sensor. Darker green cells represent decisions by the algorithm that determine a safe direction for the vehicle to travel, while following the trajctory calculated from the CNN mask and still considering data from the Lidar sensor. Darker blue cells represent navigation decisions where the Lidar sensor is prioritized over the CNN mask due to faster processing and higher accuracy of Lidar. In such cases, the vehicle will navigate based on the Lidar data, with a fixed angular direction, which is referred to in the diagram as "slightly". Red cells indicate no safe direction could be found, which means that vehicle is commanded to stop and then reverse to find a safer direction. Our sensor fusion takes both sensors as inputs to a state machine that is continuously executing as a ROS node (orange cell). Initially, a navigation target with speed and direction is determined and based on the mask from the segmentation. Before finalizing a navigation decision, the updated distance data from the Lidar is also considered according to the velocity of the initial navigation target. If the data from the Lidar confirms that there are no obstacles in that targeted direction, then the vehicle follows this direction with the appropriate speed. Otherwise when an obstacle is detected in the target direction, then the vehicle will search for a safe direction while prioritizing the Lidar's sensor data. If a safe direction still cannot be found, then the vehicle will reverse for a few seconds and then repeat the search for a safe direction once the Lidar sensor confirms that the vehicle is cleared from obstacles.

Such a pair of redundant sensors not only improves the accuracy of the vehicle perception (as mentioned earlier), but it also improves the performance of the vehicle's scan rate. When sensors can operate in parallel to each other, the amount of scans the navigation controller receives in a given time interval is the accumulation of both of those sensors added together, creating the overall scan rate. Our state machine navigates and reacts to obstacles according to the most updated data from both sensors, where each sensor updates its data independently and at its maximum rate. This implementation of parallel sensors enables a sensor fusion with performance that meets the real-time frame rate standard of 30 frames per second, because it doesn't need to wait for both sensors to send data and can operate with short delays from the CNN. Figure 19 shows the scan rate speedup from fusing lidar with our camera-based vision. This results in a system response time that is usually less than 27ms on both TX2 and AGX Xavier configurations.

VII. ANALYSIS AND DISCUSSIONS

Considering the accuracy, performance, and energy efficiency tradeoffs of our systems comparing the different CNNs, an optimal model can be correctly selected. In Figure 20, each is equally weighted and normalized for a fair comparison between models. Normalization is based on the mean and standard deviation of mIoU (accuracy), optimized forward-pass time (performance), and net power consumption (energy-efficiency).

Based on all of the statistics and results, we can recommend ENet for the mobile-edge vehicle as it achieves more optimal performance and energy-efficiency compared to the other CNNs, with a relatively minor reduction in accuracy. If the accuracy for camera-based vision needs to be improved, ERFNet would be our next recommendation

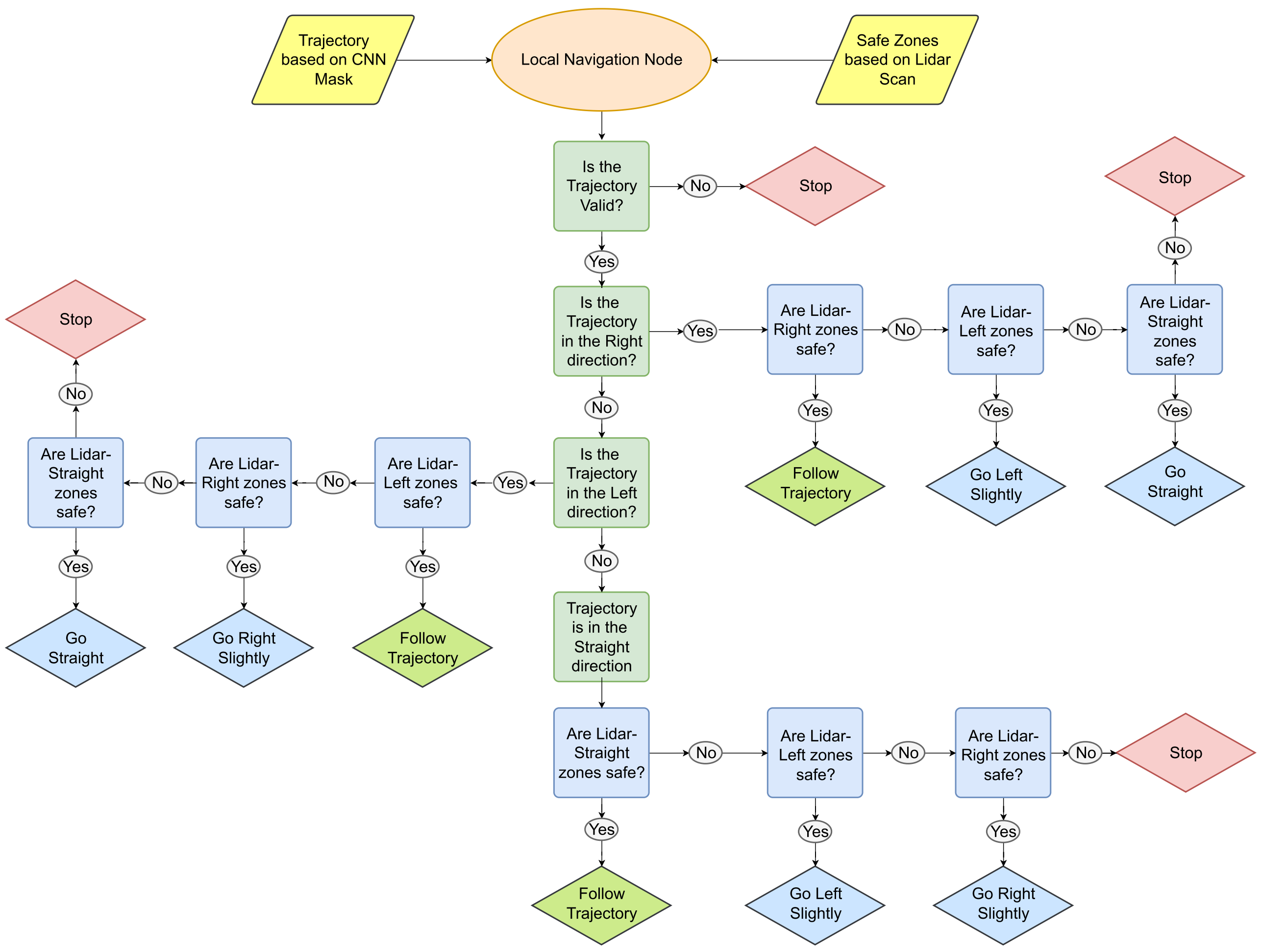
Trajectory based on CNN Mask
Local Navigation Node
Safe Zones based on Lidar Scan
Is the Trajectory Valid?
No
Stop
Yes
Is the Trajectory in the Right direction?
Are Lidar-Right zones safe?
Are Lidar-Left zones safe?
Are Lidar-Straight zones safe?
Follow Trajectory
Go Left Slightly
Go Straight
Is the Trajectory in the Left direction?
Go Right Slightly
Trajectory is in the Straight direction
Are Lidar-Straight zones safe?

FIGURE 18. Sensor fusion flow diagram.

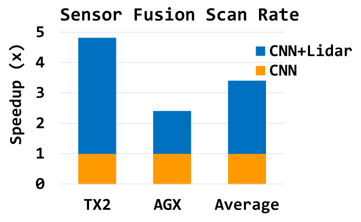

FIGURE 19. Sensor fusion scan rate speedup when fusing our camera (using ENet) with lidar.

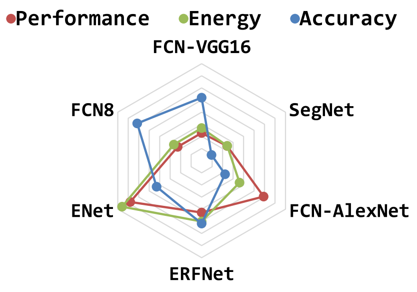

FIGURE 20. Normalized comparison between the models (points closer to the CNN name-label are better).

as long as the speed of camera-based vision can afford to be reduced by roughly 40%, which may be acceptable for slower speed robots. If accuracy is the highest priority while performance and energy efficiency can be ignored, FCN8 and FCN-VGG16 would be the best solutions of these models. Based on our qualitative assessment, we found that FCN8 (stride of 8) is better for segmenting the contours of the obstacles while FCN-VGG16 (stride of 32) is more so suited for generic clustering of obstacles, meaning that their edges and contours would not be as important. For navigation in earthquake-struck zones, FCN8 would be better than FCN-VGG16 due to the higher priority of needing to detect the irregular contours of the road cracks and other irregular types of obstacles encountered.

To further understand why some of the models do better than others and like we discussed in the earlier sections, we broke each model down at a per-layer level. The more accurate models compute over 100M parameters, although they suffer from worse performance and energy efficiency. The models originally designed for semantic segmentation

(ERFNet, ENet, SegNet) show that by widely distributing the parameters across layers/modules, they can be designed efficiently with a low number of total parameters. For instance, SegNet computes fewer parameters than the models converted to FCN because the SegNet decoder reuses max-pooling indices from the encoder when upsampling the feature maps. However, as the accuracy results showed, this technique turned out to be detrimental to SegNet's accuracy, resulting in a worse accuracy than all the other models we tested. ENet computes the fewest amount of parameters because of its heavy utilization of bottleneck modules based on dilated and asymmetric convolutions, which significantly reduces the size and execution time of each convolution, as shown in the graphs mentioned earlier. These design features in ENet resulted in it as performing better than all the other models we tested, in terms of both FPS, latency, as well as energy-efficiency, all while not suffering from significant tradeoffs in terms of the accuracy, making it an ideal model for self-navigation on mobile platforms.

VIII. CONCLUSION

In this paper, we present a mobile-based unmanned vehicle design that operates in real-time and can detect unique obstacles in earthquake-struck zones amongst other areas where road conditions are not ideal. From various field test measurement with our proof-of-concept vehicle, we observed unique differences of various semantic segmentation models and identified performance bottlenecks. With optimizations applied on computation acceleration and sensor processing, we achieved up to 5× speedup than baseline design.

In this work, we treated different types of obstacles equally for the proof-of-concept vehicular design, meaning that objects were defined as either an obstacle or non-obstacle. In future work or on a larger scale production, priority can be given to the vehicle to further decrease the risk of collision/damage. One such example would be if the platform simultaneously detects road cracks to the left and gravel (which is defined in this paper as an other ''irregular object'') to the right. In such a scenario, the platform would still steer straight and away from both obstacles, but it could also assess the risk of both objects and choose to aim its trajectory more away from the road crack (riskier obstacle) while providing itself more margin for error, as the less-risky gravel type of obstacle would not be as significant towards damaging the vehicle.

ACKNOWLEDGMENT

The work was conducted with the startup fund that Hyeran Jeon received from University of California Merced.

REFERENCES

[1] H.-H. Jebamikyous and R. Kashef, ''Autonomous vehicles perception (AVP) using deep learning: Modeling, assessment, and challenges,'' *IEEE Access*, vol. 10, pp. 10523–10535, 2022, doi: 10.1109/ACCESS.2022.3144407.

[2] R. Stricker, M. Eisenbach, M. Sesselmann, K. Debes, and H.-M. Gross, ''Improving visual road condition assessment by extensive experiments on the extended GAPs dataset,'' in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.

[3] M. Nie and K. Wang, ''Pavement distress detection based on transfer learning,'' in *Proc. 5th Int. Conf. Syst. Informat. (ICSAI)*, Nov. 2018, pp. 435–439.

[4] 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,'' Aug. 2020. *arXiv:2008.13101*.

[5] F. Yang, L. Zhang, S. Yu, D. V. Prokhorov, X. Mei, and H. Ling, ''Feature pyramid and hierarchical boosting network for pavement crack detection,'' *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 4, pp. 1525–1535, Apr. 2020.

[6] L. Zhang, F. Yang, Y. D. Zhang, and Y. J. Zhu, ''Road crack detection using deep convolutional neural network,'' in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2016, pp. 3708–3712.

[7] Q. Zou, Y. Cao, Q. Li, Q. Mao, and S. Wang, ''CrackTree: Automatic crack detection from pavement images,'' *Pattern Recognit. Lett.*, vol. 33, no. 3, pp. 227–238, Feb. 2012.

[8] Y. Shi, L. Cui, Z. Qi, F. Meng, and Z. Chen, ''Automatic road crack detection using random structured forests,'' *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 12, pp. 3434–3445, Dec. 2016.

[9] H. Oliveira and P. L. Correia, ''CrackIT—An image processing toolbox for crack detection and characterization,'' in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 798–802.

[10] J. Huyan, W. Li, S. Tighe, Z. Xu, and J. Zhai, ''CrackU-Net: A novel deep convolutional neural network for pixelwise pavement crack detection,'' *Struct. Control Health Monitor.*, vol. 27, no. 8, p. e2551, Aug. 2020.

[11] P. Pinggera, S. Ramos, S. Gehrig, U. Franke, C. Rother, and R. Mester, ''Lost and found: Detecting small road hazards for self-driving vehicles,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 1099–1106.

[12] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, ''The cityscapes dataset for semantic urban scene understanding,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 3213–3223.

[13] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, ''Augmented reality meets computer vision: Efficient data generation for urban driving scenes,'' *Int. J. Comput. Vis.*, vol. 126, no. 9, pp. 961–972, Sep. 2018.

[14] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, ''Segmentation and recognition using structure from motion point clouds,'' in *Proc. ECCV*, 2008, pp. 44–57.

[15] H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama, and H. Omata, ''Road damage detection and classification using deep neural networks with smartphone images,'' *Comput. Aided Civil Infrastruct. Eng.*, vol. 33, no. 12, pp. 1127–1141, 2018.

[16] X. Gao, T. Chen, R. Niu, and A. Plaza, ''Recognition and mapping of landslide using a fully convolutional DenseNet and influencing factors,'' *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 14, pp. 7881–7894, Aug. 2021, doi: 10.1109/JSTARS.2021.3101203.

[17] R. Ashour, M. Abdelkader, J. Dias, N. I. Almoosa, and T. Taha, ''Semantic hazard labelling and risk assessment mapping during robot exploration,'' *IEEE Access*, vol. 10, pp. 16337–16349, 2022, doi: 10.1109/ACCESS.2022.3148544.

[18] J. Ji, X. Lu, M. Luo, M. Yin, Q. Miao, and X. Liu, ''Parallel fully convolutional network for semantic segmentation,'' *IEEE Access*, vol. 9, pp. 673–682, 2021, doi: 10.1109/ACCESS.2020.3042254.

[19] A. Petrovai and S. Nedevschi, ''Semantic cameras for 360-degree environment perception in automated urban driving,'' *IEEE Trans. Intell. Transp. Syst.*, early access, Mar. 14, 2022, doi: 10.1109/TITS.2022.3156794.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, ''Densely connected convolutional networks,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[21] S. Jegou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio, ''The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jul. 2017, pp. 1175–1183, doi: 10.1109/CVPRW.2017.156.

[22] A. Valenzuela, C. Arellano, and J. E. Tapia, ''Towards an efficient segmentation algorithm for near-infrared eyes images,'' *IEEE Access*, vol. 8, pp. 171598–171607, 2020, doi: 10.1109/ACCESS.2020.3025195.

[23] E. L. Droguett, J. Tapia, C. Yanez, and R. Boroschek, ''Semantic segmentation model for crack images from concrete bridges for mobile devices,'' *Proc. Inst. Mech. Eng., O, J. Risk Rel.*, vol. 236, no. 4, pp. 570–583, 2022, doi: 10.1177/1748006X20965111.

[24] J. Li, Q. Sun, K. Chen, H. Cui, K. Huangfu, and X. Chen, ''3D large-scale point cloud semantic segmentation using optimal feature description vector network: OFDV-Net,'' *IEEE Access*, vol. 8, pp. 226285–226296, 2020, doi: 10.1109/ACCESS.2020.3044166.

[25] S. Matsuzaki, H. Masuzawa, and J. Miura, "Image-based scene recognition for robot navigation considering traversable plants and its manual annotation-free training," *IEEE Access*, vol. 10, pp. 5115–5128, 2022, doi: 10.1109/ACCESS.2022.3141594.

[26] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "ESPNetv2: A light-weight, power efficient, and general purpose convolutional neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9190–9200.

[27] I. A. Kazerouni, G. Dooly, and D. Toal, "Ghost-UNet: An asymmetric encoder–decoder architecture for semantic segmentation from scratch," *IEEE Access*, vol. 9, pp. 97457–97465, 2021, doi: 10.1109/ACCESS.2021.3094925.

[28] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "GhostNet: More features from cheap operations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1580–1589.

[29] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham, Switzerland: Springer, 2015, pp. 234–241.

[30] A. Bréhéret. (2017). *Pixel Annotation Tool*. [Online]. Available: https://github.com/abreheret/PixelAnnotationTool

[31] NVIDIA. (2017). *Jetson TX2*. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetso% n-tx2/

[32] NVIDIA. (2018). *Jetson AGX Xavier*. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetso% n-agx-xavier/

[33] M. Hua, Y. Nan, and S. Lian, "Small obstacle avoidance based on RGB-D semantic segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 886–894.

[34] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 2, Mar. 1985, pp. 500–505.

[35] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017.

[36] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and W. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Sep. 2010.

[37] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized ConvNet for real-time semantic segmentation," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 263–272, Jan. 2018.

[38] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," 2016, *arXiv:1606.02147*.

[39] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder–decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[40] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*.

[41] E. Fernandez-Moral, R. Martins, D. Wolf, and P. Rives, "A new metric for evaluating semantic segmentation: Leveraging global and contour accuracy," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 1051–1056, doi: 10.1109/IVS.2018.8500497.

[42] S. Ramos, S. Gehrig, P. Pinggera, U. Franke, and C. Rother, "Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 1025–1032.

[43] NVIDIA. (2021). *NVCaffe*. [Online]. Available: https://github.com/NVIDIA/caffe

[44] T. Sämann, K. Amende, S. Milz, C. Witt, M. Simon, and J. Petzold, "Efficient semantic segmentation for visual bird's-eye view interpretation," in *Intelligent Autonomous Systems*, M. Strand, R. Dillmann, E. Menegatti, and S. Ghidoni, Eds. Cham, Switzerland: Springer, 2019, pp. 679–688.

[45] X. Zhao, P. Sun, Z. Xu, H. Min, and H. Yu, "Fusion of 3D LiDAR and camera data for object detection in autonomous vehicle applications," *IEEE Sensors J.*, vol. 20, no. 9, pp. 4901–4913, May 2020, doi: 10.1109/JSEN.2020.2966034.

[46] Z. Zhuang, R. Li, K. Jia, Q. Wang, Y. Li, and M. Tan, "Perception-aware multi-sensor fusion for 3D LiDAR semantic segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 16280–16290.

[47] G. A. Kumar, J. H. Lee, J. Hwang, J. Park, S. H. Youn, and S. Kwon, "LiDAR and camera fusion approach for object distance estimation in self-driving vehicles," *Symmetry*, vol. 12, no. 2, p. 324, Feb. 2020, doi: 10.3390/sym12020324.

RYAN ZELEK received the B.S. degree in electrical engineering from the University of Illinois at Chicago, in 2016, and the M.S. degree in computer engineering from San Jose State University, in 2020. In 2014, he did an internship with Siemens Healthcare, as an Electrical Engineer. Since 2016, he has been working as a Customer Systems Engineer with Micron Technologies. He has been collaborating with MoCA Lab at UC Merced as an Independent Research Scientist, since 2020. His research interests include computer architecture, efficient computing, deep learning, and embedded systems.

HYERAN JEON (Member, IEEE) received the Ph.D. degree in computer engineering from the University of Southern California, in 2015. She is currently an Assistant Professor with the Department of Computer Science and Engineering, University of California Merced, Merced, CA, USA. She has industry experiences as a Systems Software Engineer and a Research Intern at Samsung Electronics, AMD Research, and IBM T.J. Watson Research Center. Her research interests include efficient computer architecture and systems in the perspectives of energy, performance, and security.