KITTI Road Segmentation

Robert Lee (V00856638)
Dept. of Electrical and Computer Engineering
University of Victoria
Victoria, Canada
robertlee@ieee.org

Ysabel Yao (V00893035)
Dept. of Electrical and Computer Engineering
University of Victoria
Victoria, Canada
xiuxiuysabel@gmail.com

***Abstract*—Lane recognition is one of the key technologies in the field of autonomous vehicles. It is widely used in assisted driving systems, lane departure warning system and vehicle collision prevention system, and it has great significance in improving traffic safety. The lane detection is susceptible to interference from external environment such as illumination variations, road occlusions and markings and vehicles on the road. This requires larger database to get a better result.**

This document describes implementing a computer vision convolutional neural network (CNN) to perform semantic segmentation of road surfaces within a driving context. To improve the model performance, data augmentation was performed using noise addition, light adjustment, contrast adjustment, etc. We used the database from KITTI which includes about 500 images in three different categories. We propose an improved network architecture, using a UNet network structure, with a ResNet18 or ResNet50 decoder. We obtained good model performance and propose future work to improve results.

Keywords—CNN, machine learning, supervised, computer vision, road semantic segmentation

I. Problem Description and Motivation

With the increasing popularity of cars, the incidence of road traffic accidents is increasing, which negatively affects the safety of people's life and property. Automatic driving cars are safer compared with manned vehicles because of a continuously updating sensor suite. They also offer potential improvements in road utilization and reduction in the occurrence of traffic jams and impaired driving. Many companies have partnered with automakers to invest in autonomous driving. Waymo, formerly Google Self-Driving Car Project, began their self-driving car project in 2009 [1]. They started by gathering information on complex environments, and advanced to learn how to navigate with pedestrians, road work, traffic lights and more since 2015. Currently, their self- driving vehicles have begun test drives on public roads without anyone in the driver's seat.

Advanced Driver Assistance System (ADAS) is an advanced technology which can understand the surrounding environment, perceive potential driving situations and improve vehicular safety [2]. This set of technologies encompass the following features: parking assistance system, blind spot detection, lane-keeping assistance, and others. The main purpose is to predict the driver's intent, warn the driver about lane departure and assist lane keeping [3]. With the deepening of the lane detection, great progress has been made in both the accuracy and speed of lane detection. However, there is a big gap with the human recognition, and there are still many technical problems that waiting to be solved in particular applications. This project will optimize the lane detection based on this problem. The objective is to use KITTI database to train model to perform road semantic segmentation. We used a UNet CNN model [4] and limited our model input to monocular RGB input images. Object detection or occlusion and other interference factors such as strong shadows are not included in this project due to time constraints.

II. Related Work

As autonomous vehicle technology is a burgeoning field, the research field has a plethora of approaches. Road detection and drivable space identification is a critical component of any auto-driving system.

Classic computer vision approaches to lane detection relies on using image processing to distinguish the lane markings. In general, the lane detection can be realized in following steps [5]: image distortion removal, applying color and gradient thresholds to the lane lines, making an aerial view by means of perspective transformation, finding hot lane lines pixels through a sliding-window mechanism, determine the left and right lines, calculate the related parameters such as lane curvature and lane center deviation and finally, drawing the related pictures and information.

Recent approaches have leveraged machine learning techniques, with deep neural networks used in many state-of-the-art solutions. However, there is a significant challenge of collecting adequate training data and manually annotating it for fully supervised learning. Not only is it dependent on time, season, rain, snow, fog, haze, glare, among others, but road surfaces vary between different cities and countries.

Thus, it is desirable for a learning method that is semi-supervised (SSL) or weakly supervised (WSL). *Han et al.* [6] proposes a method based on conditional Generative Adversarial Networks (GAN) which achieves state-of-the-art performance on the KITTI ROAD benchmark dataset. Their generator model attempts to produce realistic samples, and their discriminator model attempts to identify them from real samples. The SSL model uses a small number of labeled images in the training process. The larger number of unlabeled images are used to augment the extracted features, making it more robust to scene variations.

The authors extend the SSL method into a WSL method where road shapes are used for weak supervision. Annotations of road shapes on labeled images are used to train a network that predicts road shapes within unlabeled images. The road shapes are annotated with three labels: *branch*, *curve*, and *straight*.

The SSL method achieves the same accuracy as a fully supervised learning (FSL) method with only 400 labeled images. The WSL method achieves similar performance, with the WSL method outperforming by a small margin.

Lyu et al. proposes to use Long Short-Term-Memory (LSTM) to help reduce training and evaluation time for a typical CNN model [7]. Their network consists of a CNN-based local feature encoder, a CNN and LSTM-based feature processor, and a CNN-based output decoder [7]. The innovation is in using LSTM in the feature processor. This module is able to extract context features within a sequence of inputs while reducing the number of calculations required. Their network had 348,801 parameters, which was only 24% of the number of parameters when compared to SegNet [8], and still achieved good performance of around 90% accuracy and F1-score.

Borkar et al. [9] used inverse perspective mapping (IPM) for image transformation, which it transformed the captured images from camera perspective to a bird's eye view. In the IPM image, they got templates for narrow and wide markers were created using Gaussian kernels. Using the low-resolution Hough transformation and acquiring the correction coefficients, they find a set of lane marker candidates. Next, they were using outliner removal with random sample consensus (RANSAC) algorithm. The algorithm used to estimate parameters for fitting a mathematical model to a set of observed data that may contains outliners. Finally, a Kalman filter is used to track and smooth the estimates on the measurements. This lane detection method is based on its features. They mainly use the lane, the road and nearby texture, gradient and grey scale to split the lane.

Similarly, *Cai H et al* [10] also used the lane performance to implement their model. They state that vehicle self-localization plays critical role on safety driving assistant and automatic driving. They use a lane detection method by integrating lane shape and color features as well as using the detected lane for online vehicle position calculation. A Gaussian Statistical Color Model (G-SCM) is created for image processing to extract the region of interest. They also detected the lane and get vanishing point in the direction of the lane by utilizing the improved Hough transformation. Finally, the vehicle position can be calculated by the corresponding relation of the image coordinate system and the world coordinate system.

For different condition in weather, *Lee S et al* [11] tracked the lane and road marks in rainy day and night scene with low illumination. They establish the lane and road marked benchmark consists of about 20000 images with 17 lane no rain, rain, heavy rain and night. They used vanishing point annotation to localize the vanishing point in a road scene where parallel lanes supposedly meet. In the case of lane classes, they were using IPM to separate the sample points near vanishing points, clustering and lane regression. For the road classes, they were using grid sampling and box clustering. The VPP models that they created contains four quadrant channels which includes VP and one absence channel.

In another research [12], the authors used an image sensor process for lane detection. They used inverse perspective mapping and random sample consensus parabola fitting for lane detection, lane control by pure pursuit steering controller and classical proportional integral speed controller based on a nonholonomic kinematic model.

III. Problem Formulation

In this section, we discuss the problem search space, scope, technology stack, dataset, network architecture, and cloud training.

A. Problem Domain

We used a convolutional neural network (CNN) for our model. The problem input domain is monocular camera images of road conditions. The size is $l \times w \times 3$ individual 8-bit numbers for an RGB camera of resolution $l \times w$. Each 8-bit number has 2^8 possible combinations, which gives a total input domain size of $(2^8)^{l \times w \times 3}$. For the KITTI dataset [13], this is a total possible input domain size of $2^{8 \times 3 \times 1242 \times 375} = 2^{11,178,000}$. While the input domain of camera imagery is immense, due to the very large number of possible images, the ones that are applicable to driving situations are a much smaller subset of all possible images. Specifically, input images should have a road that extends from the vantage point onwards into the distance, and various obstacles, foliage, and vehicles in the path. We also downscaled the input images to 640 x 192.

The problem search space is the domain of all possible weights for every layer within the CNN. The number of tunable parameters was 15.8 million for the UNet based on a ResNet 18 encoder, and 37.6 million for the UNet based on a ResNet 50 encoder [4] [14].

B. Project Scope

In this project, the objective was to use a KITTI Road dataset [13] to train a road semantic segmentation convolutional neural network model. The KITTI road/lane dataset includes images and ground truths in a variety of urban and suburban environments: urban marked roads, urban unmarked road and some roads/lanes with multiple markings. We limited our model input to monocular RGB input images, downscaled to 640 x 192. We did not explicitly train the model to deal with occlusion and other interference factors such as strong shadows. Further, the model only segments the image into road surface and everything else.

We used image augmentation to increase the robustness of our model. The model does not do any segmentation of specific objects nor depth estimation, which are performed by some other models. We did not collect nor annotate our own dataset and did not modify the KITTI dataset. Since the lane ground truth data was only available for a third of the training dataset, we elected to use the road ground truth. We used transfer learning with ResNet encoder weights trained on ImageNet [15].

C. Technology Stack

We implemented our neural network using Python 3.6.8 [16], which offers robust support for TensorFlow 1.13.0 [17] and Keras 2.2.4 [18] machine learning libraries. The following libraries were used:

- *numpy*: matrix and vector operations
- *scipy*: scientific computing and optimization
- *opencv*: image processing and display
- *matplotlib*: image processing
- *tensorflow*: back end machine learning framework
- *keras*: open-source neural-network library for many popular machine learning frameworks
- *imgaug*: a powerful package for image augmentation
- *re*: regex support for identifying a model checkpoint
- *argparse*: process optional command line parameters

D. Dataset

The dataset we used is KITTI Road/Lane Detection Evaluation 2013 [13]. This dataset consists of 289 training and 290 test images, each of which has a resolution of 1242x375. The KITTI Vision datasets offer various standardized datasets for mainly vehicular applications, such as depth, object, tracking, and more.

The dataset consists of three categories of road scenes, with the train and test splits denoted by (train/test):

- *uu:* urban unmarked (98/100)
- *um:* urban marked (95/96)
- *umm:* urban multiple marked lanes (96/94)

These are combined into a final type denoted *urban*.

We augmented this data to help create a more robust model. The main transformations we apply are horizontal flip, brightness, contrast, saturation, and hue jitter. Other image augmentation techniques include rotation, Gaussian noise, and random dropout, which could be used to improve the dataset robustness [19]. This increased the data we had available to train and allowed the model to learn to be robust to these types of changes. We used an 80/20 random split for training and validation, and the separate provided test set.

E. Network Architecture

We used a standard fully convolutional UNet [4] structure that receives an RGB colour image as input and generates a same-size semantic segmentation map as output (see Figure 1). The structure of the network is an encoder-decoder network with skip connections between various feature levels of the encoder to the decoder. This enables the network to combine information from both deep abstract features and local, high-resolution information to generate the final output. The encoder section uses a configurable ResNet18 or ResNet50 model [14]. Both models were tested.

Figure 1: Network architecture

F. Cloud Training

Since training CNNs requires high performance GPUs with support for NVIDIA CUDA [20], we augmented local CPU-based training (Macs do not offer NVIDIA GPUs) with cloud training. After investigating Microsoft Azure, Amazon Web Services (AWS), and Google Colab, we decided to use Google Colab [21]. Training on Colab resulted in over a tenfold reduction in training time.

IV. Evaluation

We begin this section with a brief evaluation of the strengths of a residual network UNet fully-convolutional neural network. We then discuss the performance metrics that are used in road semantic segmentation. We next describe the loss functions that are used to train the neural network. Finally, we evaluate the performance of our model, both in terms of ResNet18- and ResNet50-based models.

A. Strengths of Chosen Network Architecture

The chosen UNet structure is a proven architecture for extracting both deep abstract features and local features. It consists of three sections: the contraction, bottleneck, and expansion. The downsampling contraction section, known as the *encoder*, extracts feature information from the input by decreasing in spatial dimensions but increasing in feature dimensions. The bottleneck is where we have a compact representation of the input. The upsampling expansion section, known as the *decoder*, reduces the image in feature dimension while increasing the spatial dimensions. It uses *skip* connections that allow it to tap into the same-sized output of the contraction section, which allows it to use higher-level locality features in its upsampling. This is a strong architecture for image segmentation, as we must assign each pixel a class. Thus, our output must use the high-resolution information from the input image to obtain a good prediction.

The encoder was chosen to be a ResNet model because it allows us to train a deep neural network and reduces the risk of the vanishing gradient problem. The key breakthrough is in the eponymous residual connections (see Figure 2), which allow gradient information to have an alternate path to flow through.

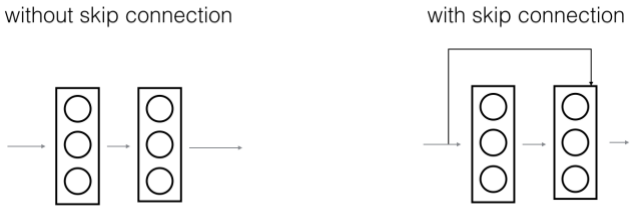

Figure 2: Residual block [22]

B. Performance Metric Types

For the driving dataset problem domain, there are many common evaluation metrics. Often, a simple pixel-wise comparison fails to account for small variations in the model that may be acceptable to the problem domain, or others that have a high negative impact. The authors propose a transform

into Bird's Eye View (BEV) space [23]. This is stronger than a pixel-wise evaluation, as such evaluation measures are weak at determining the safe *driving corridor* and *ego-lane boundaries*. This transform also has the strength of simplifying calculations such as lane length, corridor width, and boundary position.

Given the assumption that detection results are in the form of confidence maps or binary maps, we can transform between the image domain and BEV domain. There are a number of evaluators that many existing solutions are evaluated with, and we used those as well to compare our results.

Pixel-based metrics evaluate the model using per-pixel comparisons between the predicted and ground truth image. It is easier to calculate, and we will likely rely on this initially. However, there are a number of limitations with this approach, as it measures the quality of all pixels. This may not be adequate if we are using this information to inform a vehicle's decisions. Thus, this only gives a rough indication of performance.

Behavior-based metrics evaluate the behavior of a vehicle that may follow the results of the model. For example, we can evaluate the model on how well a theoretical vehicle can follow the detected lane. We can determine a *driving corridor* which may be adequate at city speeds and evaluate the vehicle behavior. This is much more complicated, as there are numerous steering maneuvers a vehicle can perform, which will greatly increase the number of possible behaviors. Since writing these metrics in the TensorFlow and Keras Backend graph language is non-trivial, we elected to use pixel-based metrics for the first implementation that are widely documented.

C. Training Loss

For the loss function of the model, we used Dice loss [24] [25]. Dice loss is a statistic developed to determine the similarity between two samples. The Dice coefficient is:

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

In this equation, p_i and g_i represent pairs of corresponding pixel values of prediction and ground truth. Dice loss considers the loss function both locally and globally, which is critical for high accuracy. Dice loss can be use with predicted probabilities directly instead of thresholding and converting them into binary mask [26].

D. Evaluation of Output

To evaluate the performance of the model, we used a pixel-based F_{max} measure, which is calculated as follows:

$$F\text{-measure} = (1+\beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2\,\text{precision} + \text{recall}}$$

$$F_{max} = \underset{\tau}{\text{argmax}}\, F\text{-measure}$$

where τ is a classification threshold.

To get the accuracy of the results, first, the precision and recall can be calculated by these equations [27]:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Precision can be shown by in a confusion matrix as:

		Predicted: Negative	Predicted: Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

True Positive + False Positive = Total Predicted Positive

Figure 3: Precision in Confusion Matrix

Therefore, the denominator part of the precision is total predicted positive. It means how precise/accurate your model is out of those predicted positive and how many of them are actual positive. When the costs of False Positive is high, precision is a good measure to determine. Precision is the fraction of relevant instances among the retrieved instances while recall is the fraction of the total amount of relevant instances that were actually retrieved. The formulation can be shown as follow:

$$Recall = \frac{True\ Positive}{TruePositive + False\ Negative}$$

The recall can be shown in a confusion matrix as:

		Predicted: Negative	Predicted: Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

True Positive + False Negative = Actual Positive

Figure 4: Recall in Confusion Matrix

As we can see in the confusion matrix, recall actually calculates how many of the actual positives model capture through it as True Positive. When there is a high cost associated with False Negative, Recall is the model metric.

F1 Score is for when want to find a balance between Precision and Recall. It is the harmonic mean of Precision and Recall and gives a better measure of the incorrectly classified cases than the Accuracy Metric. Accuracy is used when the True Positives and True Negatives are more important while F1 score is used when the False Negatives and False Positive are crucial [28].

In our evaluation function, we use the ground truth image and model prediction to calculate the F1 score.

V. Results

The following pictures shows the example of the evaluation result. The model input is shown in Figure 5. The generated segmentation is marked as blue overlayed on the original image and is shown in Figure 6.

Figure 5: Input image

Figure 6: UM Overlayed result

The original ground truth and predicted images are marked the road as blue and others are combined by other color, in this case, we only considered the channel 0 (blue channel, in BGR pixel order) and generated the images as shown in Figure 7 and Figure 8:

Figure 7: UM ground truth

Figure 8: UM Predicted Model

By calculating the F1 score and accuracy using the methods in the previous section, the best case for F1 score is around 99.1% and the worst case is around 91.3%. The results are summarized in Table 1.

*Table 1: Model performance evaluation using F1 and Accuracy. **um:** urban marked; **umm:** urban multiple marked lanes; **uu:** urban unmarked*

Image Type	Best / Worst	ResNet 18 Encoder		ResNet 50 Encoder	
		F1 (%)	*Accu (%)*	*F1*	*Accu (%)*
UM	*Best*	97.938	97.643	98.222	97.852
	Worst	95.311	94.561	94.078	93.688
UMM	*Best*	96.910	96.443	96.743	95.612
	Worst	95.765	95.198	94.652	92.317
UU	*Best*	98.432	98.180	99.066	98.639
	Worst	93.953	92.310	92.494	91.326

Using Tensorboard, the logs shown in Figure 9 and Figure 10 were generated during training of the ResNet50-based model. Accuracy refers to the default keras accuracy metric, and loss refers to the Dice loss metric. The validation set is prefixed by *val_*.

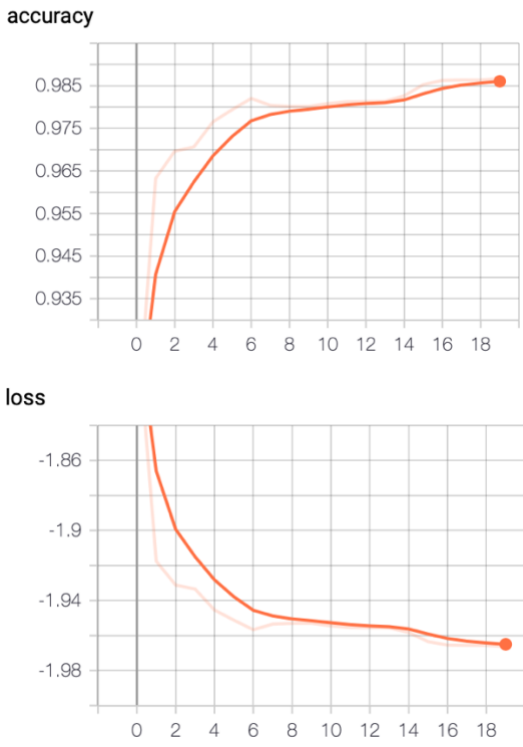

Figure 9: Accuracy and Dice loss of training set

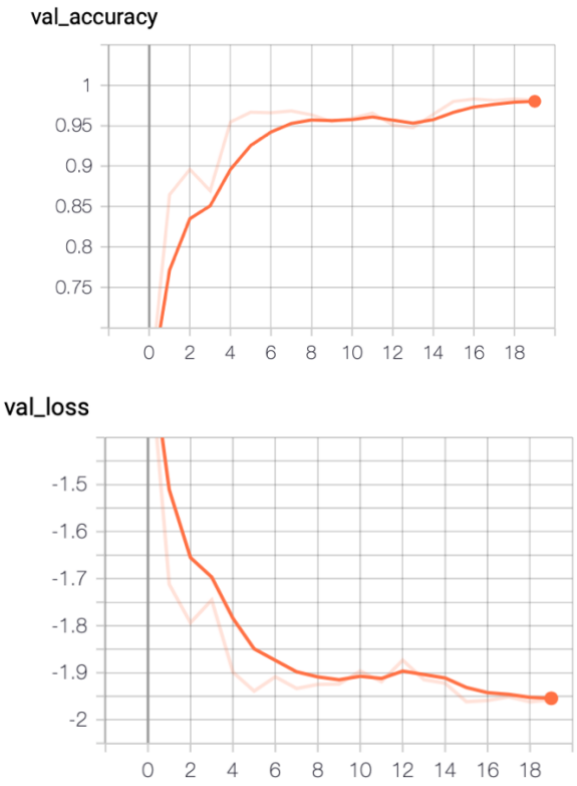

Figure 10: Accuracy and Dice loss of validation set

The model performs well in a variety of conditions:

Figure 11: From top to bottom: irregular shadows, cracks and road curvature, multiple cars, lane widening

The model suffers in a few situation types, shown below:

Figure 12: From top to bottom: extra wide initial road area, railroad tracks, high contrast shadow regions and uncertain sidewalk boundaries, road surface split by a median

VI. Discussion

For the evaluation function, the F1 score in best case was 99.1% and accuracy is 98.6%; in worse case, the F1 score was 92.5% and accuracy is 91.3%. We also analyzed the two different encoders: ResNet18 and ResNet50. In the UU case, ResNet50 performed better than ResNet18. However, ResNet50 did not perform well in some complicated situations; thus, it sometimes performed worse.

We note that the trained model is not overfitting, since the training and validation accuracy are both relatively monotonously increasing with respect to epochs. In addition, the training and validation loss function is also steadily decreasing with respect to epochs. This is seen in Figure 9 and Figure 10.

We observe that the model performs best when the road surface is clearly distinguished from surroundings. It is especially strong when there are distinct lane markings. This is likely because the model can extract the edges as an indicator of potential road areas.

As shown in Figure 11, the model is robust in many different situations. As shown in Figure 12, the model is weak when there are misleading lines that are visually similar to road surfaces, such as the train tracks. It also has difficulty when there is a high contrast shadow region, or where the road is split by a median. The model seems to favour segmentations that are not split on the lower section of the image. It also prefers connected road regions. This is reasonable, since real-world roads will be connected. However, this results in road surfaces that are split from the very bottom of the image being incorrectly segmented, such as the bottom image in Figure 12. However, this may be tolerable, because the car cannot move into that lane anyway.

In Table 1, we see the model has both the best and worst performance on urban unmarked images. This class of images has the greatest variation because urban environments have irregularly shaped roads, many pedestrians and parked vehicle occlusions, and highly varied surrounding obstacles, among others. It is reasonable that the model performs the worst in this class.

Tighter spread in performance is seen on the urban multiple marked lines dataset. This is likely because the model has learned to use lane markings to determine road surfaces, and multiple parallel lines will strengthen that prediction. In addition, wider roads requiring multiple lane markings are often found in the outskirts of the city, where the roads are naturally straighter and have fewer random objects and occlusions.

We note that the ResNet 50-based model had both better and worse performance. This is reasonable, since deeper networks with more trainable parameters may perform better if we can provide a larger dataset. Since the given dataset only contains hundreds of images, there may not be enough data to train the deeper model to achieve consistently better performance. Deeper models are also more likely to suffer from the gradient vanishing problem, which may affect the propagation of weight updates.

Multiple challenges were encountered while training the model. The biggest challenge was an incorrect concatenation in the decoder. The final layer was returned as a concatenation of multiple scale outputs, which likely jumbled the network results, causing it to output extremely large positive and negative values. Further, the final layer of that decoder had parts of the model that were visible, without a softmax or sigmoid function that can cap the output values. After fixing it with final output layer of softmax activation, the model results were between the range of [0,1] as expected.

Another bug was encountered in the segmentation ground truth and was fixed by producing a binary one-hot segmentation for each pixel from the ground truth data.

Challenges were also encountered in implementing non-standard loss functions, such as birds' eye view (BEV), in the TensorFlow and Keras language. This is because the available graph operations are limited, and workarounds must be found to implement some math operations. We decided to revert to a pixel-based Dice loss function.

The dataset also posed challenges. The ground truth data for the testing dataset was not available for users to download. Instead, they provide a server where results can be uploaded and evaluated. This is likely because they want to prevent cheating in the benchmark. However, this made it difficult to evaluate the performance of the testing dataset. This was solved by visually inspecting the test output, and calculating the model performance metrics on training data, knowing that this will overestimate the model performance. The test output was similar to the training output, so this did not greatly affect the results.

Another dataset issue was the lack of available driving corridor segmentation ground truth data. This was only available for a third of the total training data. Thus, we changed our plan from performing lane segmentation to road segmentation. If lane segmentation is still desired, we can use weights pre-trained on the road dataset to fine tune with the limited lane marking dataset.

Finally, issues were encountered with the various command line arguments that we programmed. This was solved by testing various combinations of parameters and inspecting and refactoring possible branches.

VII. Conclusion

In this project, we performed road segmentation by training on the dataset from KITTI which includes over 500 images. We used a standard UNet model architecture with a ResNet 18 or ResNet 50 encoder and trained the model in Google Colab. We used a UNet structure to extract both deep abstract features and local features. For the evaluation part, we used a pixel-based metric and chose dice loss for our loss function. We calculated the F1 score and accuracy which is 99% and 98% for our best result. Finally, we concluded that there are limitations in the training dataset and our model, and we need more data for more accuracy.

VIII. Future Work

While the results of the model were strong, there remains work that can improve the results. The major item that will likely greatly improve results is implementing stronger loss functions based on the birds' eye view metric discussed in *Section IV.B: Performance Metric Types*. This was not implemented in the first iteration of this model because the calculations require extensive debugging to translate it from normal Python into a language suitable for Keras Backend and TensorFlow. The operations that are available are limited, and not all calculations have a suitable mapping. By improving the loss function, the model can better tune the network parameters during backpropagation. A behaviour-based loss function also improves the results in a driving context.

The current output has a jagged appearance because Upsample2D blocks were used in the decoder. This is a simple scaling of the image, and while it is efficient, it results in artifacts in the output. This is especially noticeable in the last layer, where a Upsample2D of 4x is used, and produces the scaling pixilation artifacts seen in the output. This block can be extended to multiple Upsample2D of 2x separated by convolution layers, which will likely reduce pixilation.

All of the Upsample2D blocks can also be changed to use a Conv2DTranspose. This also upsamples, but it uses a convolution kernel that is trainable, which can improve model performance.

The decoder is fairly shallow, especially compared to the ResNet50 encoder backbone depth. This may mean we are losing details in the upsampling. We can increase the depth of the decoder to increase the number of trainable parameters, which may improve results. However, this may also lead to a decrease in performance, so this should be compared to the current result.

Finally, we can explore using better Keras metrics to evaluate the training of the model. For example, Mean Intersection-Over-Union may be used. This computes the IOU for each segmentation class and averages them and is a better metric for segmentation problems.

References

[1] "We're building the World's Most Experienced Driver," *Waymo*. [Online]. Available: https://waymo.com/. [Accessed: 31-Aug-2020].

[2] V. S. Bisen, "What Is ADAS Technology And How It Works In Car For Safe Driving?," *Medium*, 13-Dec-2019. [Online]. Available: https://medium.com/vsinghbisen/what-is-adas-technology-and-how-it-works-in-car-for-safe-driving-62395883310b. [Accessed: 31-Aug-2020].

[3] C. Y. Kuo, Y. R. Lu, and S. M. Yang, "On the Image Sensor Processing for Lane Detection and Control in Vehicle Lane Keeping Systems," *Sensors (Basel, Switzerland)*, 08-Apr-2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6479783/. [Accessed: 31-Aug-2020].

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. UNet: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[5] Nachiet Tankale, "Finding Lane Line- Simple Pipeline for Lane Detection", *Medium*, [Online]. Avaliable: https://towardsdatascience.com/finding-lane-lines-simple-pipeline-for-lane-detection-d02b62e7572b [Accessed Jul. 20,2020]

[6] Xaiofeng Han, Jiangfeng Lu, Chunxia Zhao, Shaodi You, Hongdong Li. “Semisupervised and Weakly Supervised Road Detection Based on Generative Adversarial Networks”, *IEEE*, vol.25, no.4, April 2018, [Online].

[7] Yechen Lyu, Lin Bai, Xinming Huang. “Road Segmentation Using CNN and Distributed LSTM”. Worcester Polytechnic Institude, Mar 2019. [Online]. Avaliable: https://arxiv.org/pdf/1808.04450.pdf [Accessed Jul. 20,2020]

[8] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[9] A. Borkar, M. Hayes and M. T. Smith, "A Novel Lane Detection System With Efficient Ground Truth Generation," in IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 1, pp. 365-374, March 2012, doi: 10.1109/TITS.2011.2173196.

[10] H. Cai, Z. Hu, G. Huang and D. Zhu, "Robust road lane detection from shape and color feature fusion for vehicle self-localization," 2017 4th International Conference on Transportation Information and Safety (ICTIS), Banff, AB, 2017, pp. 1009-1014, https://ieeexplore.ieee.org/document/8047893

[11] S. Lee, et al., “VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition,” *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[12] C.Y.Kuo, Y.R.Lu, S.M.Yang. *Multidisciplinary Digital Publishing Institute*, April 2019, [Online]. Avaliable: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6479783/. [Accessed Jul. 20,2020]

[13] “Road/Lane DetectionEvalustion 2013”. *Karlsrue Institute of Technology*. [Online]. Avaliable: http://www.cvlibs.net/datasets/kitti/eval_road.php. [Accessed Jul. 20, 2020]

[14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Dec. 2015.

[15] Rami Karim. “Illustrated: 10 CNN Architetures”, *Medium*. [Online]. Avaliable: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d [Accessed Jul. 20,2020]

[16] “Python 3.6.8”. *Python*. [Online. Avaliable: https://www.python.org/downloads/release/python-368/ [Accessed Jul. 20, 2020]

[17] “TensorFlow White Papers: TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/about/bib. [Accessed: 01-Feb-2020].

[18] K. Team, “Simple. Flexible. Powerful.,” *Keras*. [Online]. Available: https://keras.io/. [Accessed: 31-Aug-2020].

[19] Alxandra Deis. “Data Augmentation for Deep Learning”, *Medium*. [Online]. Avaliable: https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9 [Accessed Jul. 20, 2020]

[20] “CUDA GPUs,” *NVIDIA Developer*, 17-Aug-2020. [Online]. Available: https://developer.nvidia.com/cuda-gpus. [Accessed: 31-Aug-2020].

[21] “Google Colaboratory,” *Google*. [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynb. [Accessed: 31-Aug-2020].

[22] P. Dwivedi, “Understanding and Coding a ResNet in Keras,” *Medium*, 27-Mar-2019. [Online]. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33. [Accessed: 31-Aug-2020].

[23] Jannik Fritsch, Tobias Kuhnl, Andreas Geiger. “A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms”.[Online]. Avaliable: http://www.cvlibs.net/publications/Fritsch2013ITSC.pdf [Accessed Jul. 20, 2020]

[24] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, “Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations,” *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support Lecture Notes in Computer Science*, pp. 240–248, 2017.

[25] S. Du, “Understanding Dice Loss for Crisp Boundary Detection,” *Medium*, 29-Feb-2020. [Online]. Available: https://medium.com/ai-salon/understanding-dice-loss-for-crisp-boundary-detection-bb30c2e5f62b. [Accessed: 31-Aug-2020].

[26] Jeremy Jordan, “An overview of semantic image segmentation.,” *Jeremy Jordan*, 12-Jan-2019. [Online]. Available: https://www.jeremyjordan.me/semantic-segmentation/. [Accessed: 31-Aug-2020].

[27] K. P. Shung, “Accuracy, Precision, Recall or F1?,” *Medium*, 10-Apr-2020. [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. [Accessed: 31-Aug-2020].

[28] P. Huilgol, “Accuracy vs. F1-Score,” *Medium*, 24-Aug-2019. [Online]. Available: https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2. [Accessed: 31-Aug-2020].