Semantic segmentation on driving images for applications in autonomous driving technologies

Karina Tsang, Tiffany Shih, Fidelia Nawar

Problem Statement:

Autonomous driving has gained popularity in recent years with the aim of reducing driver errors and improving road safety. Successful automation requires a series of detection of the vehicle's surroundings, starting from general segmentation of the environment to accurate detection of nearby vehicles and traffic signs. This project aims to compare the accuracy and efficiency of two different machine learning models in image segmentation and object classification in a driving environment.

Objective:

Perform semantic segmentation on a driving image by accurately classifying the main parts of a driving image by the following categories: drivable area and background.

Approach/ Methodology:

In order to meet our objective of segmenting drivable area vs. background of images, the image segmentation technique we utilized was semantic segmentation for our models. Semantic segmentation is the technique for classifying each pixel of an image to a class label (e.g. driving area or background). This differs from instance segmentation, where each unique object or area of the image is classified and categorized.

Our approach for our models was to classify the regions of the image as drivable area and background. This would be performed by using attributes such as pixel intensity values (in grayscale), color, and location of each cluster within the image. Additional attributes that may be considered would be the time of day and weather while the image was taken. The specific models we used for this project are Fully Convolutional Network (FCN) and Mask R-CNN. Below are descriptions of each model type:

FCN: The first model used for the semantic segmentation portion of the project was FCN, Fully Convolutional Network. As shown in Figure 1, the FCN architecture includes 5 pooling layers, each includes a number of convolution layers and a pooling layer [1]. The outputs of these pooling layers are then transposed in order to merge with the output of a previous layer. FCN-8 is used in this exercise.

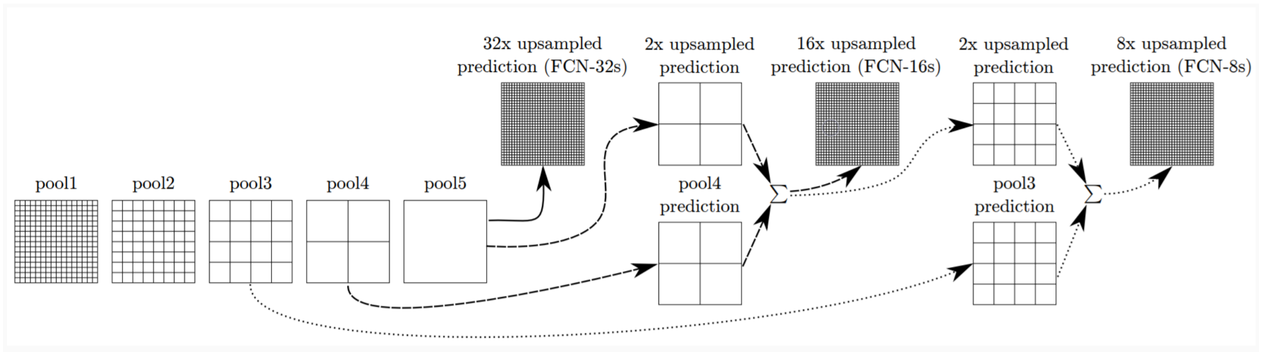


Figure 1: FCN Architecture

VGG 16 architecture was used to first create the first five pooling layers, as shown in Figure 2 below. The later merging was additionally created to replace the three dense layers, and outputs of Pool3, Pool4, and Pool5 were extracted for merging purposes.

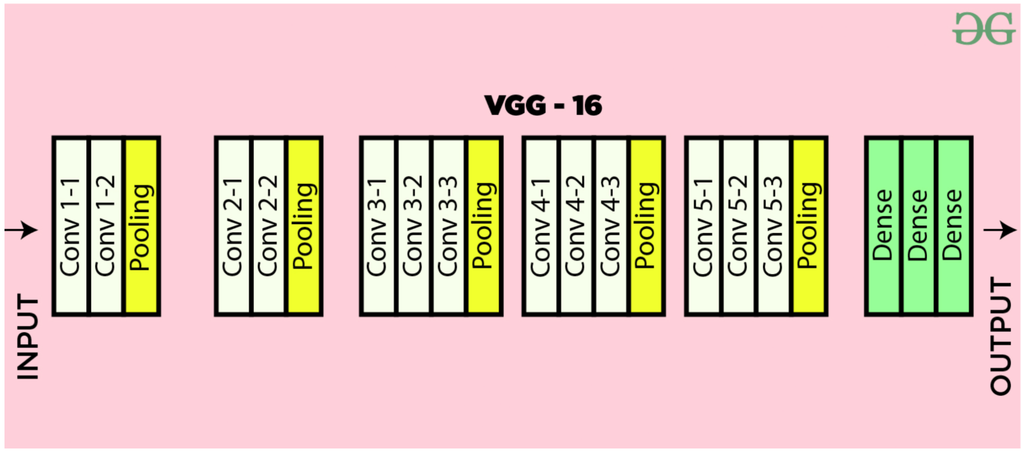


Figure 2: VGG 16 Architecture [1, 2]

Mask R-CNN: Classification modeling methods were performed by Mask R-CNN, which is useful for instance segmentation. Mask R-CNN was developed by the Facebook AI Research team and is considered one of the industry leading model frameworks for image segmentation [3].

This method is an extension of Fast-RCNN (which has two outputs for each candidate object) and works by adding a branch for predicting an object mask (region of interest) in parallel with the existing branch for bounding box recognition. The key feature of Mask R-CNN is pixel-to-pixel alignment. It adopts the same two-stage procedure of Fast R-CNN, where it first creates a Regional Proposal Network (a neural network that proposes multiple objects that are available within a particular image). In the second stage, in parallel to predicting the class and box offset, Mask R-CNN also outputs a binary mask for each region of interest pooling. Figure 3 below shows the Mask R-CNN framework for instance segmentation:

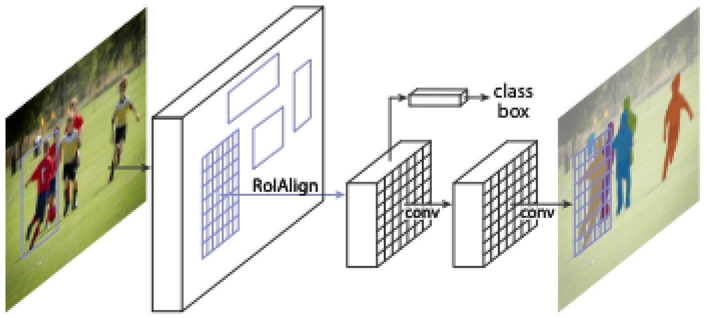


Figure 3: Mask R-CNN Framework for Instance Segmentation [4]

This project applied the Mask R-CNN algorithm to detect drivable areas in the image annotated through polygons vertices. In contrast to the FCN's implementation of the VGG-16 architecture, the Mask R-CNN utilizes the RESNET-101 or RESNET-50 architecture for feature extraction. The outputs of the Mask R-CNN model for an input image are the detected object classes, their bounding box coordinates, and an object mask showing which pixels on the image correspond to the object within the bounding box.

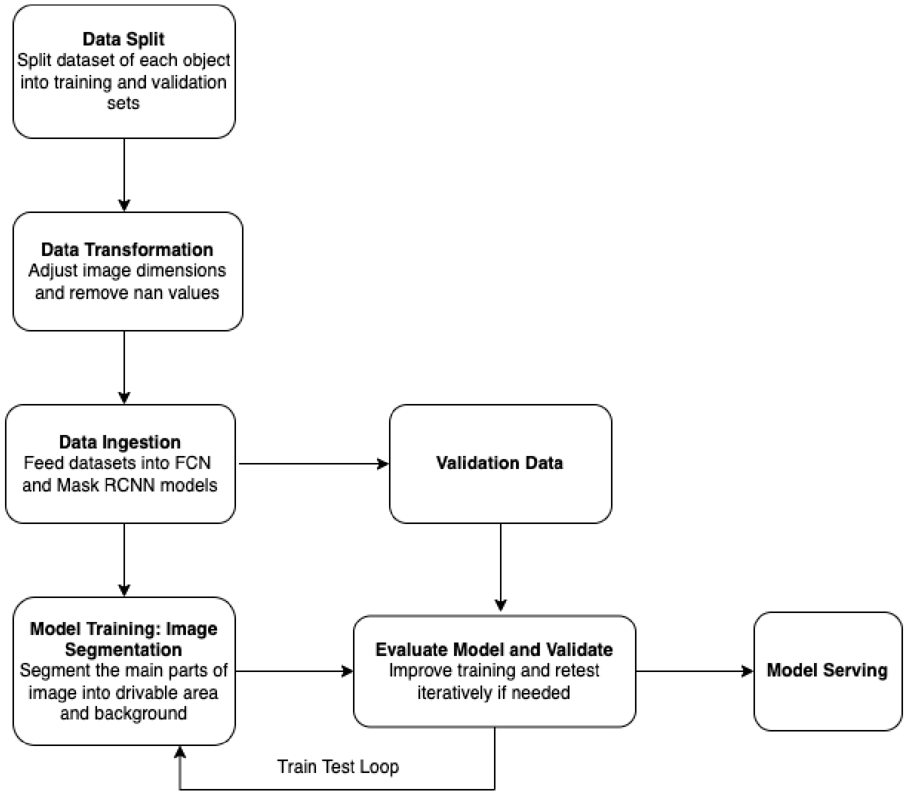
Model Attributes:

Both FCN and Mask R-CNN rely on image and pixel features to generate model attributes for model weights. Below are examples of image and pixel features that are used as inputs for the model.

- Pixel values: Used as an attribute to determine contrast between an object and background. Threshold value may be set to differentiate between the drivable area and background. Used for region-based segmentation.
- Filtered values: Kernel filters applied on pixel values can allow for edge detection.
- Neighboring clusters: Relationship between pixels and their neighboring pixel values.

Block Diagram

Figure 4 below depicts the block diagram of our model development process.



Data Split
Split dataset of each object into training and validation sets
Data Transformation
Adjust image dimensions and remove nan values
Data Ingestion
Feed datasets into FCN and Mask RCNN models
Validation Data
Model Training: Image Segmentation
Segment the main parts of image into drivable area and background
Evaluate Model and Validate
Improve training and retest iteratively if needed
Model Serving
Train Test Loop

Figure 4: Block Diagram of Model Development Process

Datasets

The dataset used for this project comes from Berkeley DeepDrive (https://www.bdd100k.com/) [5]. The data was collected from the San Francisco Bay Area and in New York CIty. The dataset itself included videos, 100k images (frames at the 10th second in the videos), 10k images (for segmentation), labels (annotations of road objects), video attributes (such as weather, scene, timeofday in JSON), and lane markings, and drivable area masks/colormaps to distinguish the drivable area on a road. There are also different label formats for object detection (10 classes evaluated such as 0: pedestrian, 1: rider, 3: car, etc) and instance segmentation for infrastructure such as sidewalks, buildings, walls, and drivable areas.

Dataset Preparation

For each of the models, we cleaned data by extracting the number of features of each object, such as number of roads, sidewalks, buildings, etc, and then extracted the labels. We also only ran the model on images from the training and validation datasets filtered by the following conditions: weather == "clear", scene == "highway", and timeofday == "daytime".

After cleaning the datasets through this filtering process, we cleaned the training, validation, and test labels for each model (described in Experiment section below). We then split the training set into training/validation subsets. There were a total of 3575 images in the dataset with the conditions for weather == "clear", scene == "highway", and timeofday == "daytime". We performed a 90/10 split of the training and validation data, which resulted in 3218 training images and 357 validation images.

What is considered success/ failure?

Success is correctly identifying the regions of the image per the appropriate class (drivable area and background), and correctly classifying the objects in the image. The percentage threshold criteria used for success is 50%.

Evaluation Parameters

The models are expected to produce output images that will identify the drivable area, alternative drivable area, and the background. The output images will then be compared to the already segmented images in the test set (see Figure 5). Each pixel will be scored 1 for being correctly segmented and 0 for incorrectly segmented, which aggregates into an accuracy score for each image. The aggregation of the accuracy scores for all images provided by a model will be used for evaluation.

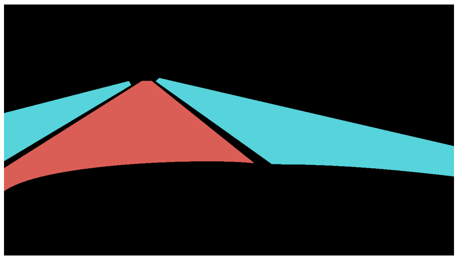


Figure 5. Example test set image for image segmentation exercise.
Note that blue areas (alternate drivable area) and red areas (drivable area) shown on the image will both be classified as drivable area for this study.

The evaluation parameters used to compare models' performance for image segmentation and object detection & classification will include:

- **Mean Accuracy:** Mean value of # correctly identified pixels/number of pixels for all images in validation dataset
- **Mean Precision:** Mean value of precision for each validation image. Precision for each image is the proportion of positive identifications of pixels that are actually correct (correctly identified pixels as drivable area / identified pixels as drivable area from the model).
- **Mean Recall:** Mean value of recall for each validation image. Recall for each image is the proportion of correct pixels that are identified correctly (correctly identified pixels as drivable area / all actual, labeled pixels as drivable area)

- **Mean F1-score:** Mean value of F1 scores for each validation image. F1 score is the harmonic mean of precision and recall.

Experiments

FCN

Data Ingestion:
To prepare the data for model ingestion, the image will need to be imported as numpy arrays using the cv2 (OpenCV) package. As the fcn model requires transposing of feature maps for merging purposes, square images would be more convenient. Therefore, we also resize the image into the shape of (128,128).

Data Cleaning:
For the FCN model, we had to ensure that the training and validation datasets had no nan values that would prevent the model from being able to fit and evaluate. We were unable to simply remove the nan values because that creates a dimensionality reduction, so we imputed the nan values with the average of the other pixel array elements. We also had to perform checks to ensure there were no infinite values in the data set. There was also an issue of exploding gradients that resulted in a large update to network weights during training. To combat this issue, we performed gradient clipping so that the gradients have a certain “maximum allowed model update” before fitting.

Model Tuning Parameters:
As the dataset is large and model fitting requires a significant amount of time, there are limitations in experimenting with different ranges of epochs and batch size. We have adopted mini-batch gradient descent, testing two sets of parameters: 1) epoch = 5, batch_size = 100 and 2) epoch = 20, batch_size = 4 where the latter shows better outcomes.

Evaluation metrics:
To evaluate the results with metrics such as accuracy, precision, recall, etc., we have to compare the pixels between the true mask and the predicted mask, and determine whether the pixels in the predicted mask are categorized correctly into the drivable area or the background. In order to do so, we have to change the pixel value from the 3 channel RGH values into 0 or 1, representing background and drivable area respectively. In order to do so, the predicted and true masks are converted into grayscale, and classified into 0 or 1 based on a threshold that separates the drivable and background areas. Then, the classification of each pixel is compared between the true and predicted masks to get TP, TN, FP, FN and eventually the evaluation metrics.

Mask R-CNN

Environment Set-Up:
Installation of proper packages to run the model was a complex process, as it required the proper combination of package versions, hardware compatibility, and sufficient memory to run the model successfully. Proper combination of packages that work with Tensorflow 2.x as latest numpy packages are not compatible with all Tensorflow releases. In addition, Tensorflow 2.x installation on Mac M1 chip was attempted, but later determined to be unfeasible as the combination of packages required to work with the Mask R-CNN model was not immediately compatible particularly given the memory limitations. As a result, we shifted to setting up the environment on Google Cloud Platform, leveraging the Deep Learning Virtual Machine (VM) Image to set up the environment. We also utilized Google Cloud Storage to store the images and enable sharing of resources stored on the Cloud for running our projects without taking up local memory. Lastly, through experimentation, we discovered the proper minimum GPU and CPUs required to train the model in a reasonable amount of time (which in our case we considered as roughly 2 hours), which were the Tesla T4 (16GB) GPU [6] and a minimum of 30 GB RAM via 8 vCPUs.

Dataset Cleaning:
For the specific case of the mask r-cnn model, polygon vertices are provided as input to the model for semantic classification of pixels on the image. However, some of the polygon vertices included in the json label files contained values that were greater than the pixel dimension of the image. This was corrected by checking each polygon vertex and, if they were greater than image dimensions, limiting them to maximum image dimensions of 1280 x 720.

Model Development:
The model framework used was based on the Mask R-CNN model open source library [7]. We used pre-trained weights that the model developed based on the COCO dataset as a starting point for our model inputs. The COCO dataset itself is a large-scale object detection, segmentation, and captioning dataset used to develop, train, and compare models.

We then adapted the code from the Mask R-CNN model framework to our dataset by updating the number and types of classes to 2 (one background, one for drivable area), and updating the format of the training and validation label annotations to be compatible with the model input [8]. The annotated mask label format provided by the BDD dataset was generated in Scalabel Format, which is different from the expected Mask R-CNN model mask input format.

Model Tuning:

After adapting the code from the model framework, we then adjusted hyperparameters for the model to optimize performance. During this time, we trained on a few images only to evaluate time required to train vs. computational resources. Generally, we found that there was a tradeoff in the model tuning parameters for maximizing accuracy, minimizing training time, and minimizing computational resources and cost. The types of parameters we tuned for our model included the following:

- Images per GPU: Tuning considerations depends on GPU memory available for use. We used just 1 as we only use one GPU with 16GB memory (Tesla T4 Core)
- Number of epochs: Number of passes through training data that the model runs through to update the model weights
- Steps per Epoch: Equivalent to training dataset size / batch size. In this case, we set this to 1 consistently to allow each batch to run through the entire training dataset size
- Validation Steps: Number of validation steps run at end of each training epoch. Improves accuracy of validation stats but slows down training. Trading off between number of epochs and validation steps for accuracy and timing
- Backbone Architecture: Two options were available for use: RESNET50 and RESNET101. These are the standard convolutional neural networks serving as feature extractor. The numbers are associated with the number of convolutional layers in the architecture. For cases where the classes aren't as easily distinguishable, RESNET101 would likely perform better than RESNET101; however, is slower to train and may overfit. Both of these options were explored in this project, and RESNET101 was found to perform better.
- Detection Minimum Confidence: Minimum confidence required to display. Values between 0.5 to 0.9 were explored. Too high a value would result in little to no pixels classified as drivable areas, whereas too low a value would result in performance that is not more accurate than a random coin flip. Therefore, a value of 0.7 was ultimately used.

Evaluation Metrics:

To calculate the evaluation metrics, the predicted mask output was first generated by the model output then converted to usable format. The output format of mask r-cnn consists of an array of dimension 1280x720xN, where N corresponds to the number of objects detected. Because in this project we are measuring the performance of semantic segmentation and not instance segmentation, we performed a sum of the predicted mask arrays across axis=2 (the 3rd axis) in order to generate a single predicted mask array consisting of each pixel classified as drivable area or background. We then converted it to bool and back to int type, which allows for pixels that have value of 0 for all object-level masks to be classified as background and pixels that have values > 0 across any of the object masks to be classified as drivable area. From there, we compared the pixels with the true mask values to calculate accuracy, precision, recall, and F1-score for each validation image. The True Positives, False Positives, and False Negatives are specified as follows:

- True Positive: Both predicted and ground truth classification of a given pixel are drivable area
- False Positive: Predicted classification of a given pixel is drivable area, while ground truth is background
- False Negative: Predicted classification of a given pixel is background, while ground truth is drivable area

The mean values for accuracy, precision, recall, and F1-score were then taken across all validation images and reported as part of results below for comparison with the FCN model results.

Results

Model evaluation was performed across the 357 validation images for comparison between the two models. Table 1 provides the results of the evaluation parameters calculated by FCN and Mask R-CNN models based on the validation dataset. Equations for each of the parameters were calculated as follows:

1. **Mean Accuracy:** $\frac{1}{N}\sum_{i=1}^{N}\frac{Number\ of\ correctly\ classified\ pixels\ in\ ith\ image}{1280\times720}$, where N represents the total number of images in the validation dataset (which is 357 images).
2. **Precision:** $\frac{True\ Positives}{True\ Positives + False\ Positives}$, where
    a. True Positives represents the total number of pixels in the validation dataset that were correctly categorized as drivable area (i.e. ground truth for drivable area pixels matches the model's classified results for drivable area), and
    b. False Positives represents the total number of pixels in the validation dataset that the model incorrectly categorized as drivable area
3. **Recall:** $\frac{True\ Positives}{True\ Positives + False\ Negatives}$, where
    a. False Negatives represents the total number of pixels in the validation dataset that the model incorrectly categorized as background pixels
4. **F1-Score:** $2\times\frac{precision \times recall}{precision + recall}$

Table 1: Comparison of Evaluation Parameters for FCN and Mask R-CNN Models

Parameter	FCN	Mask R-CNN
Mean Accuracy	0.3137	0.5174
Precision	0.0980	0.0166
Recall	0.0886	0.0002
F1-Score	0.0829	0.0004

Figure 6 below depicts a comparison of the binary cross-entropy losses calculated for both FCN and Mask R-CNN architectures. The losses for Mask R-CNN start much higher and although they do show an eventual decrease, it shows oscillations with a local maximum around 17 epochs. Part of this reason is because the Mask R-CNN de-couples the mask predictions (boundaries of each unique object) and the class prediction as separate losses that are summed, whereas FCN combines the mask-prediction and class-prediction on a per-pixel basis to a single loss value for the image [9, 10]. However, the oscillations seen on the Mask R-CNN loss is indicative that the model has not yet converged. In contrast, the right plot in Figure 6 shows the FCN Loss vs. Epochs plot demonstrating a smooth decrease and approaching convergence.

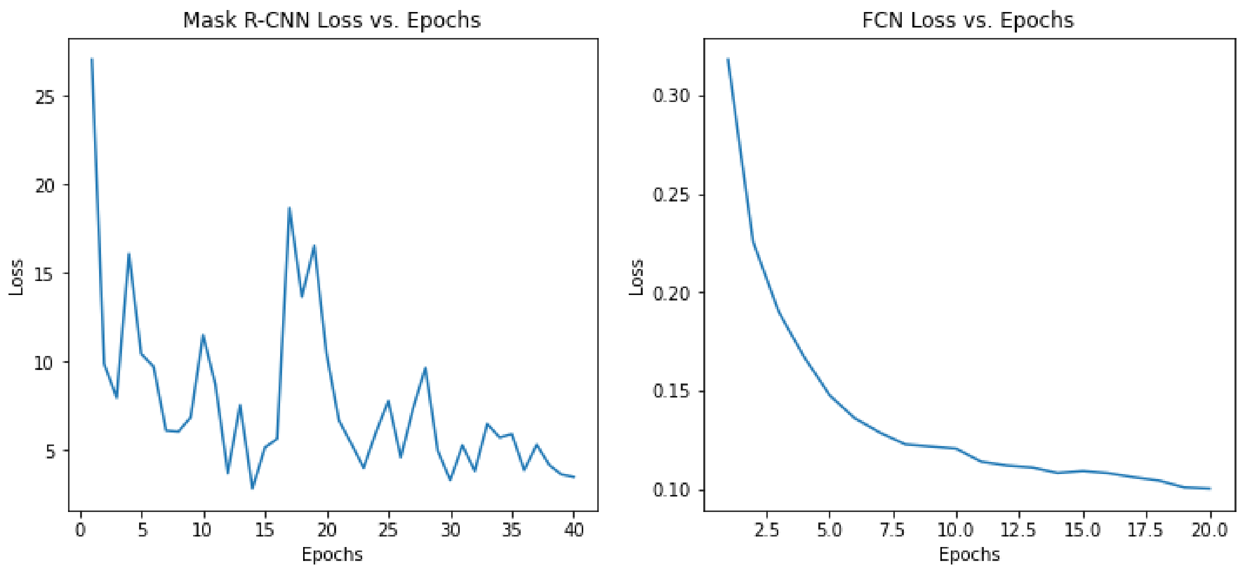


Figure 6: Loss vs. Number of Epochs for Mask R-CNN and FCN Models

Tests/Graphs/Discussions

The accuracies of the FCN and Mask R-CNN are both higher than the success criteria established. However, accuracy is not a good indicator of model performance for semantic segmentation because the majority of pixels consist of the background of the image rather than drivable area [11]. As a result, even if the model classified all pixels as background, the accuracy would still be greater than 50%. Figure 7 and Figure 8 below depict a sample FCN and Mask R-CNN model outputs for the same input image. As can be seen, the majority of the image is background (shown in the true mask).

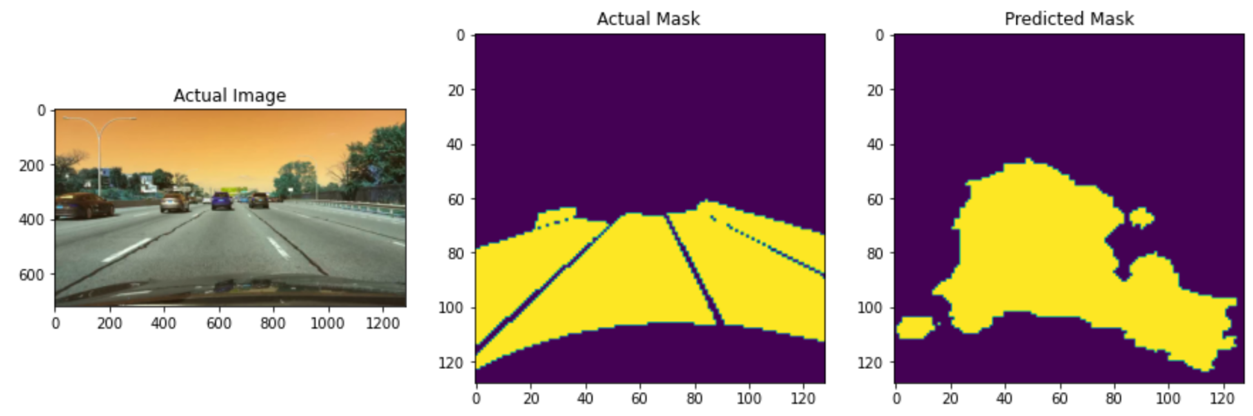


Figure 7: FCN True Mask vs. Predicted Mask Output for Sample Image 09486e3a-538126e6.jpg

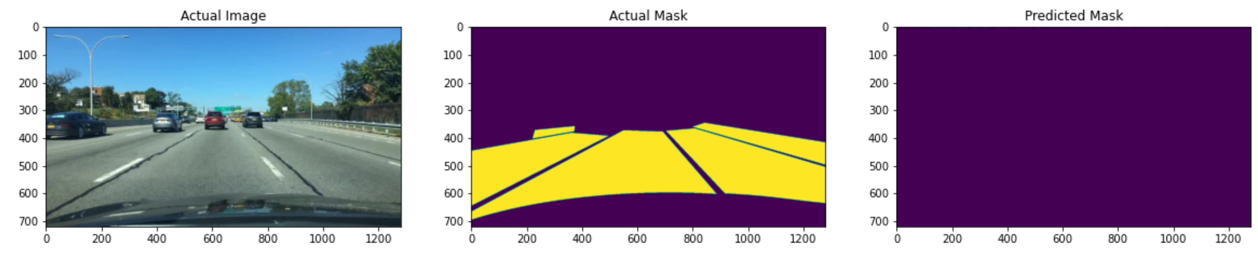


Figure 8: Mask R-CNN True Mask vs. Predicted Mask Output for Sample Image 09486e3a-538126e6.jpg

Constraints

Model Constraints:
The constraints we applied in our model were based on the conditions of the image. As specified in the Dataset Preparation section, we filtered our training and validation datasets to clear weather, highway scenes, and daytime conditions only. This means that our model performance and evaluation is constrained to only those types of conditions.

Hardware Constraints:
When running the Mask R-CNN model, the primary constraint we encountered was with computing power. Given the number of images and the complexity of the mask R-CNN model architecture, local CPU memory was insufficient for processing the images and running the model timely. In addition, running on CPU alone resulted in significant memory utilization and required much longer training times, compared to running on GPU. In order to minimize the cost to run the model, only 1 GPU was used for training purposes, as the number of images in our training dataset and the resolution of the images did not require more than 1 GPU.

Standards

Traditional image segmentation techniques can be divided into 5 main categories as shown in Figure 9 [12]. These techniques are traditionally used for digital image processing. The methods employed by these techniques include evaluating local and neighboring pixel features, gradients across pixels, and convolving an image with kernel filtering to enhance or identify specific features, edges, or objects within an image.

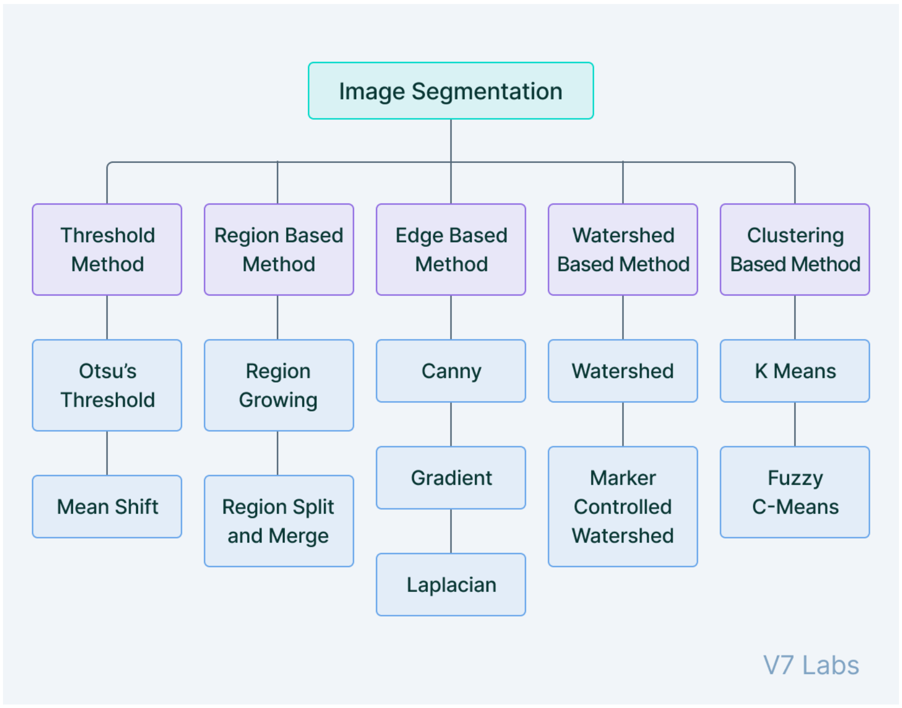


Figure 9: Traditional Image Segmentation Techniques [12]

In addition to FCN and Mask R-CNN, several other deep-learning techniques for image segmentation have emerged in the last decade such as U-Net [13, 14, 15]. U-Net is one of the most popular architectures used in biomedical image segmentation of biological microscopy images. It uses data augmentation techniques to learn from the available annotated images. U-Net architecture consists of two parts: a contracting part and a symmetric expanding path, for capturing context and enabling precise localization respectively.

CNN architectures used to construct these models are:

- VGG-16
    - VGG-16 is a VGGNet model that supports 16 layers with image input size of 224x224 and can be useful for object detection. This model includes deconvolution and pooling layers employed to identify pixel-wise label information for each class. The convolutional layers that are pretrained from this model can

    then be used to stack ReNet layers which can extract generic local features to perform image segmentation.

- RESNET50
    - RESNET 50 is 50 layers deep and can classify images into 1000 object categories. This network has learned rich feature representations for a wide range of images, also with input sizes of 224x224. Since this model is pretrained (previously trained on a dataset and contains the weights and biases that represent the features of whichever dataset it was trained on), learned features are easily transferable to different data.
- RESNET101
    - In comparison to RESNET 50, RESNET101 is 101 layers deep and is pre-trained on more than a million images from the ImageNet database. The network can also classify images into 1000 object categories. The network consists of a set of max-pooling and convolution layers to identify pixel-wise class labels and prediction of the mask.

Model Comparison

Based on the design, architecture, and intended purpose for FCN and Mask R-CNN models, an optimized FCN model should theoretically perform similarly as an optimized Mask R-CNN model in semantic segmentation of drivable areas on an image. The intended use case of Mask R-CNN differs from that of FCN because Mask R-CNN decouples the mask and class prediction. Because Mask R-CNN's outputs include a different mask for each unique object identified, Mask R-CNN is intended for use cases for counting the number of instances that different classes of objects are detected within an image. In comparison, FCN model architecture is not built to identify the number of objects of the same class in the image. For this particular application of the two models, however, the results do not provide a clear advantage of one versus the other. The accuracy of FCN appears worse than that of Mask R-CNN, but the loss, precision, recall, and f1-score all demonstrate that this implementation of FCN for this use case is higher performing compared to Mask R-CNN. Given this result compared to the theoretical performance of FCN and Mask R-CNN, this points to the potential for further optimization of both models.

Limitations of the Study

One of the significant limitations of the study is that we are filtering by images that were only taken on a clear day on the highway during the daytime. This limits the performance of our model to only be accurate on images with these specific conditions, even though there are many other driving conditions that we may typically run into.

In addition, the dataset utilized images collected from the San Francisco Bay Area and the New York Metropolitan Area. This does not provide sufficient diversity in the types of road images collected. Factors that are limited by these locations include road color and materials, climate, and objects in the background. One can imagine that the types of road conditions that are encountered in other countries would deviate significantly from those seen in the Bay Area and New York. As a result, the models generated through this study are limited to use in these locations and are likely to be significantly less effective and accurate if used to classify images from other locations around the world.

Future Work

In our original proposal, we planned to perform image segmentation in addition to object detection. In the future, we hope to implement object detection to identify objects in images such as driving lanes, vehicles, pedestrians, traffic lights/signs, buildings, etc.

One way to improve our model would be to expand the filter conditions of weather, time of day, and scene to include more varied driving environments, lighting from the different times of day, and potentially more obscure weather conditions. Training the model on more varied conditions like this would allow us to make predictions on a larger subset of data.

Another future objective would be for a given image, to identify the driveable area that the driver may drive towards. For example, in a two-direction road, identify the region that is the opposing/incoming traffic vs. the direction that the car is driving. This would involve drawing a border around the region that is in the same direction that the car is driving.

References

1. Prakash, Abhinav. “Semantic Segmentation By Implementing FCN”. Kaggle 2021. https://www.kaggle.com/code/abhinavsp0730/semantic-segmentation-by-implementing-fcn/notebook
2. “VGG-16 | CNN model”. Geeks for Geeks, 2022. https://www.geeksforgeeks.org/vgg-16-cnn-model/
3. He, Kaiming, et. al. “Mask R-CNN”. arXiv:1703.06870v3. https://doi.org/10.48550/arXiv.1703.06870
4. “Mask R-CNN”, Papers With Code, 2022. https://paperswithcode.com/method/mask-r-cnn
5. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. 2021 ETH VIS Group. https://www.bdd100k.com/
6. NVIDIA T4 TENSOR CORE GPU Datasheet. 2019 NVIDIA Corporation. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf
7. Mask_RCNN 2.0, Matterport. 2017, Github Repository, https://github.com/matterport/Mask_RCNN.git
8. Singla, Aarohi. Mask-RCNN-on-Custom-Dataset-2classes. 2021, Github Repository, https://github.com/AarohiSingla/Mask-RCNN-on-Custom-Dataset-2classes-

9. Krinski, Bruno & Ruiz, Daniel & Machado, Guilherme & Todt, Eduardo. (2019). Masking Salient Object Detection, a Mask Region-based Convolutional Neural Network Analysis for Segmentation of Salient Objects. https://www.researchgate.net/publication/335908007_Masking_Salient_Object_Detection_a_Mask_Region-based_Convolutional_Neural_Network_Analysis_for_Segmentation_of_Salient_Objects
10. Brownlee, Jason. “A Gentle Introduction to Cross-Entropy for Machine Learning”. 2021 Machine Learning Mastery. https://machinelearningmastery.com/cross-entropy-for-machine-learning/
11. Tiu, Ekin. “Metrics to Evaluate your Semantic Segmentation Model”. 2019 Towards Data Science. https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2
12. Bandyopadhyay, Hmrishav. “A Gentle Introduction to Image Segmentation for Machine Learning”. 2022 V7Labs. https://www.v7labs.com/blog/image-segmentation-guide#traditional-segmentation
13. Bianco, Simone, et. al. “Benchmark Analysis of Representative Deep Neural Network Architectures”. 2017, IEEE Access. https://arxiv.org/pdf/1810.00736.pdf
14. Dwivedi, Priya. “Semantic Segmentation — Popular Architectures”. 2019 Towards Data Science. https://towardsdatascience.com/semantic-segmentation-popular-architectures-dff0a75f39d0
15. Sultana, Farhana, et al. “Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey”. arXiv:2001.04074v3. https://arxiv.org/pdf/2001.04074.pdf