Advanced Methods for Image Processing

Kevin Nava,Abubakar Qahir

Abstract

This project focuses on developing an image segmentation model for street images under two weather conditions: sunny and rainy. The dataset consists of video frames and their corresponding segmentation masks, with 3779 images for both frames and masks in the sunny condition and 3642 images in the rainy condition. The rainy images present challenges due to noise caused by rain, resulting in poor segmentation quality. The goal of this project is to design a classifier from scratch, without pre-trained models, to segment these frames effectively. This will involve applying data preprocessing techniques, including denoising, generative models, and advanced segmentation methods, to improve the model's performance on both sunny and rainy frames. The project aims to address challenges such as noisy data and poor segmentation in rainy conditions, ultimately enhancing segmentation accuracy.

Introduction

Image segmentation plays a crucial role in computer vision, particularly in applications that involve street scene analysis, such as autonomous driving, urban planning, and surveillance. Accurate segmentation is essential for understanding the environment and making decisions based on the content of images. However, real-world challenges, such as varying weather conditions, can severely affect the quality of segmentation results.

The dataset used in this project consists of video frames and their corresponding segmentation masks, representing various elements of the street scene, such as roads, vehicles, pedestrians, and static objects. While the sunny images provide relatively clear segmentation data, the rainy images are affected by noise, leading to poor segmentation performance. This highlights the difficulty of building a reliable segmentation model for real-world scenarios where weather conditions can introduce significant variability in image quality.

To address this challenge, the project aims to build a deep learning-based classifier from scratch, without relying on pre-trained models. The classifier will be trained on the provided dataset, with the goal of accurately segmenting street images, regardless of the weather condition. In addition to developing the classifier, the project incorporates data preprocessing techniques, such as denoising, generative models, and advanced segmentation methods, to mitigate the effects of noisy data and improve the overall model performance.

Since there are 34 categories according to their definition class, here is the reference table:

Class Name	ID	Has Instance	Segmentation Color
Unlabeled	0	False	(0, 0, 0)
Static	4	False	(0, 0, 0)
Dynamic	5	False	(111, 74, 0)
Ground	6	False	(81, 0, 81)
Road	7	False	(128, 64, 128)
Sidewalk	8	False	(244, 35, 232)
Parking	9	False	(250, 170, 160)
Rail track	10	False	(230, 150, 140)
Building	11	False	(70, 70, 70)
Wall	12	False	(102, 102, 156)
Fence	13	False	(190, 153, 153)
Guard rail	14	False	(180, 165, 180)
Bridge	15	False	(150, 100, 100)
Tunnel	16	False	(150, 120, 90)
Pole	17	False	(153, 153, 153)
Pole group	18	False	(153, 153, 153)
Traffic light	19	False	(250, 170, 30)
Traffic sign	20	False	(220, 220, 0)
Vegetation	21	False	(107, 142, 35)
Terrain	22	False	(152, 251, 152)
Sky	23	False	(70, 130, 180)
Person	24	True	(220, 20, 60)
Rider	25	True	(255, 0, 0)
Car	26	True	(0, 0, 142)
Truck	27	True	(0, 0, 70)
Bus	28	True	(0, 60, 100)
Caravan	29	True	(0, 0, 90)
Trailer	30	True	(0, 0, 110)
Train	31	True	(0, 80, 100)
Motorcycle	32	True	(0, 0, 230)
Bicycle	33	True	(119, 11, 32)

Table 1: Class Definition Table

Figure 1: Samplings of frames and segmentation from DATASET

0.1 Workflow of the Code

The workflow is divided into three files:

- **preprocessing.py** loads and processes the segmentation masks, mapping colors to class IDs, converting them to grayscale, and applying a median filter for denoising. The processed masks are saved for training.
- **train.py** defines and trains the U-Net model, using the preprocessed data and a data generator to load batches. It employs early stopping and checkpointing to optimize training.
- **test.py** evaluates the trained model on the test set, compares predictions with true masks, and visualizes the results for further analysis.

Data Processing

In this project, the data preprocessing pipeline involves the following steps:

- **Loading and Mapping Segmentation Masks**: The segmentation masks are loaded from specified directories. Each pixel's color in the mask is mapped to a class ID using a predefined dictionary, converting the color-coded masks into grayscale, where each pixel represents a class ID.
- **Grayscale Mask Conversion**: After mapping the color values to class IDs, the masks are converted into a grayscale format. Each pixel in the mask is assigned its corresponding class ID based on the color mapping. This results in a 2D mask where the pixel values range from 0 to 33, representing the 34 different classes. This format is essential for training the model since it provides a numeric representation of each class.

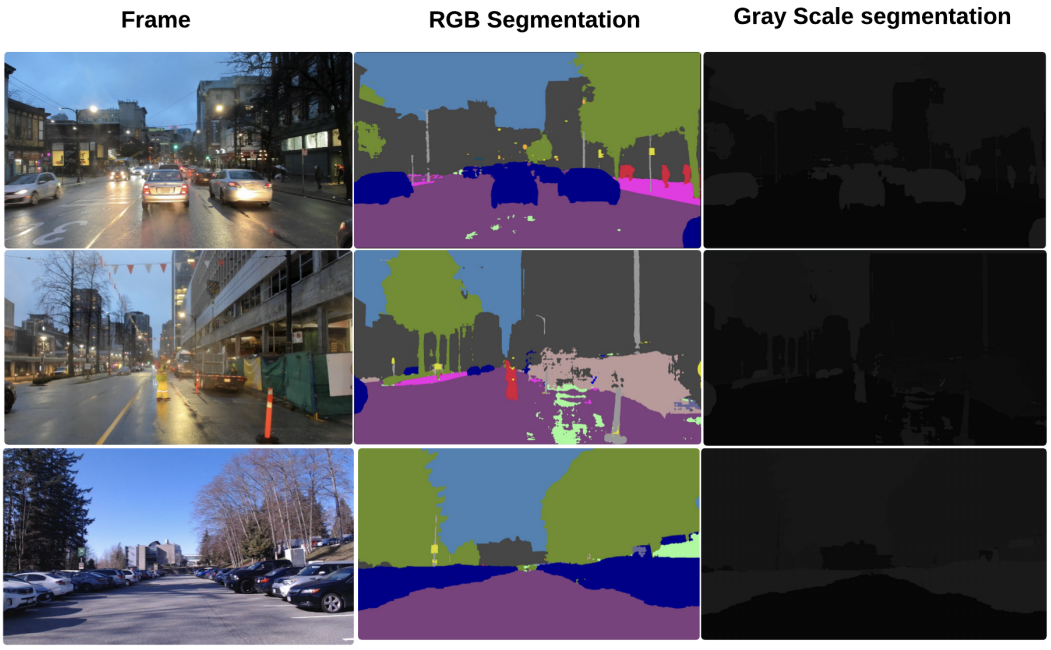

Figure 2: rgb to grayscale segmentation

- **Denoising**: A median filter is applied to reduce noise, especially in rainy images where noise can affect segmentation quality.The median filter works by replacing each pixel's value with the median value of the pixels in its neighborhood, effectively reducing small-scale noise.

- **Saving Processed Masks**: The denoised masks are saved to a new directory for use in training.

0.1.1 Convolutional Network U-NET

U-Net is a convolutional neural network designed for semantic segmentation, featuring an encoder-decoder structure. The encoder extracts hierarchical features from the input image through convolutional layers and max-pooling, reducing spatial resolution. The decoder upsamples these features and reconstructs the image using skip connections from the encoder, preserving fine details. The final output is a pixel-wise classification map, generated through a convolutional layer with softmax activation. The model is trained with the Adam optimizer and categorical cross-entropy loss, with accuracy as the metric, although other segmentation-specific metrics like IoU or Dice score are also common.

Figure 3: Workflow of U-NET

0.2 Model Architecture

1. Encoder (Compression Path)

The encoder part of the U-Net is responsible for extracting important features from the input image while progressively reducing its spatial dimensions. This is achieved through a series of convolutional layers followed by max-pooling layers. The convolutions help capture spatial hierarchies in the image, while max-pooling reduces the image size and helps the model focus on the most important features.

Block	Details
Block 1	2 convolutions (64 filters, 3×3, ReLU), followed by MaxPooling2D.
Block 2	2 convolutions (128 filters, 3×3, ReLU), followed by MaxPooling2D.
Block 3	2 convolutions (256 filters, 3×3, ReLU), followed by MaxPooling2D.
Block 4	2 convolutions (512 filters, 3×3, ReLU), followed by MaxPooling2D.

Table 2: Encoder Architecture

2. Bottleneck

The bottleneck is the central part of the U-Net, connecting the encoder to the decoder. It captures the most abstract features of the image through deep convolutional layers. This part of the network processes the lowest resolution feature maps and allows the model to understand global context before moving on to the decoder.

Block	Details
Block 5	2 convolutions (1024 filters, 3×3, ReLU), without any pooling.

Table 3: Bottleneck Architecture

3. Decoder (Expansion Path)

The decoder part of the U-Net model reconstructs the image by upsampling the feature maps and combining them with high-resolution information from the encoder through skip connections. This allows the model to retain fine details while increasing the spatial resolution of the image. Each block in the decoder is composed of upsampling followed by concatenation with features from the corresponding encoder block and convolution layers to refine the features.

Block	Details
Block 6	UpSampling2D to double the spatial resolution, concatenates with Block 4, 2 convolutions (512 filters).
Block 7	UpSampling2D, concatenates with Block 3, 2 convolutions (256 filters).
Block 8	UpSampling2D, concatenates with Block 2, 2 convolutions (128 filters).
Block 9	UpSampling2D, concatenates with Block 1, 2 convolutions (64 filters).

Table 4: Decoder Architecture

4.Output Layer

The output layer uses a **Conv2D** layer with a 1×1 kernel to generate pixel-wise classifications. **Softmax activation** is applied to produce a probability map for each pixel, suitable for multi-class segmentation.

5.Model Compilation

The model is compiled with the **Adam optimizer** and **Categorical Cross-Entropy** loss function for multi-class segmentation. **Accuracy** is used as the evaluation metric, though **IoU** or **Dice score** are often more suitable for segmentation tasks.

Denoising

1. Purpose of Denoising

Denoising is crucial for improving the quality of segmentation masks, especially under challenging conditions such as rain. Rain can introduce significant noise into the segmentation masks due to raindrops, which can make it difficult for the model to differentiate between meaningful features (e.g., roads, vehicles, pedestrians) and irrelevant noise (e.g., raindrops). By applying denoising techniques, the model can focus on the relevant features and ignore the environmental noise, leading to better performance.

2. Applying the Median Filter

The median filter is applied to the grayscale segmentation masks to reduce noise. This filter replaces each pixel with the median value of its neighboring pixels within a defined window. It effectively removes small-scale noise, such as that caused by raindrops, while preserving the larger, more important structures like roads and vehicles. This step is particularly important in rainy conditions where noise might otherwise interfere with the model's ability to detect meaningful features.

The median filter works by replacing each pixel value in the image with the median value of its neighboring pixels within a defined window. The filter operates as follows:

For a given pixel, the median filter looks at the values of the pixels in its neighborhood, sorts them, and replaces the central pixel with the median value of the sorted

neighborhood. Mathematically, for a pixel p in a 3x3 window, the new value p' is given by:

$$p' = \text{median}(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9)$$

Where $p_1, p_2, \ldots, p_9$ are the values of the surrounding pixels in the window. This filter effectively smooths out small-scale noise (like raindrops), while preserving the larger structures, such as the outlines of roads or vehicles, that are important for segmentation.

3. Improving Model Accuracy

By removing small-scale noise, denoising helps the model focus on the key features of the image, such as the road structure, vehicles, and pedestrians, ultimately improving segmentation accuracy. Without denoising, the model could mistakenly classify noise as part of these features, leading to poor performance. Therefore, denoising ensures that the model is trained on cleaner, more accurate data, allowing it to make more precise predictions, particularly when tested on images taken in adverse conditions like rain.

Denoising with Deep Image Prior

Denoising with Deep Image Prior (DIP) is a technique that utilizes a deep neural network to denoise images without the need for paired clean and noisy training data. The method optimizes a randomly initialized convolutional neural network (CNN), which is trained solely on the noisy image. The network learns to generate a clean image by fitting the noisy input. The key aspect of DIP is its use of a loss function that minimizes the pixel-wise difference between the denoised image and the noisy input, typically using L2 loss. The architecture itself acts as a prior, ensuring that the network denoises effectively without overfitting to the noise. This approach leverages the natural image priors embedded in the network's structure, making it a powerful tool for image denoising, even in cases where traditional methods fail.

Loss Function:

$$\mathcal{L} = \|\hat{I} - I_{\text{noisy}}\|_2^2$$

where $\hat{I}$ is the predicted clean image and I_{noisy} is the noisy input image.

Figure 4: Deep Prior result for 5000 epoch for single image

Denoising with BM3D

BM3D (Block-Matching and 3D filtering) is a highly effective image denoising algorithm that works by grouping similar image patches into 3D blocks and applying collaborative filtering. It operates in two main steps: first, similar patches are grouped, and then these 3D blocks are denoised using a transform-based method, such as wavelet or DCT, followed by collaborative filtering. BM3D is particularly well-suited for Gaussian noise and effectively preserves image details while reducing noise.

Formula for BM3D Denoising:

$$\hat{I} = \mathcal{T}^{-1}\left(\sum_{k} \mathcal{F}_k \cdot \mathcal{B}_k\right)$$

where $\hat{I}$ is the denoised image, $\mathcal{T}$ is the inverse transform, $\mathcal{F}_k$ is the filtering operator, and $\mathcal{B}_k$ is the 3D block.

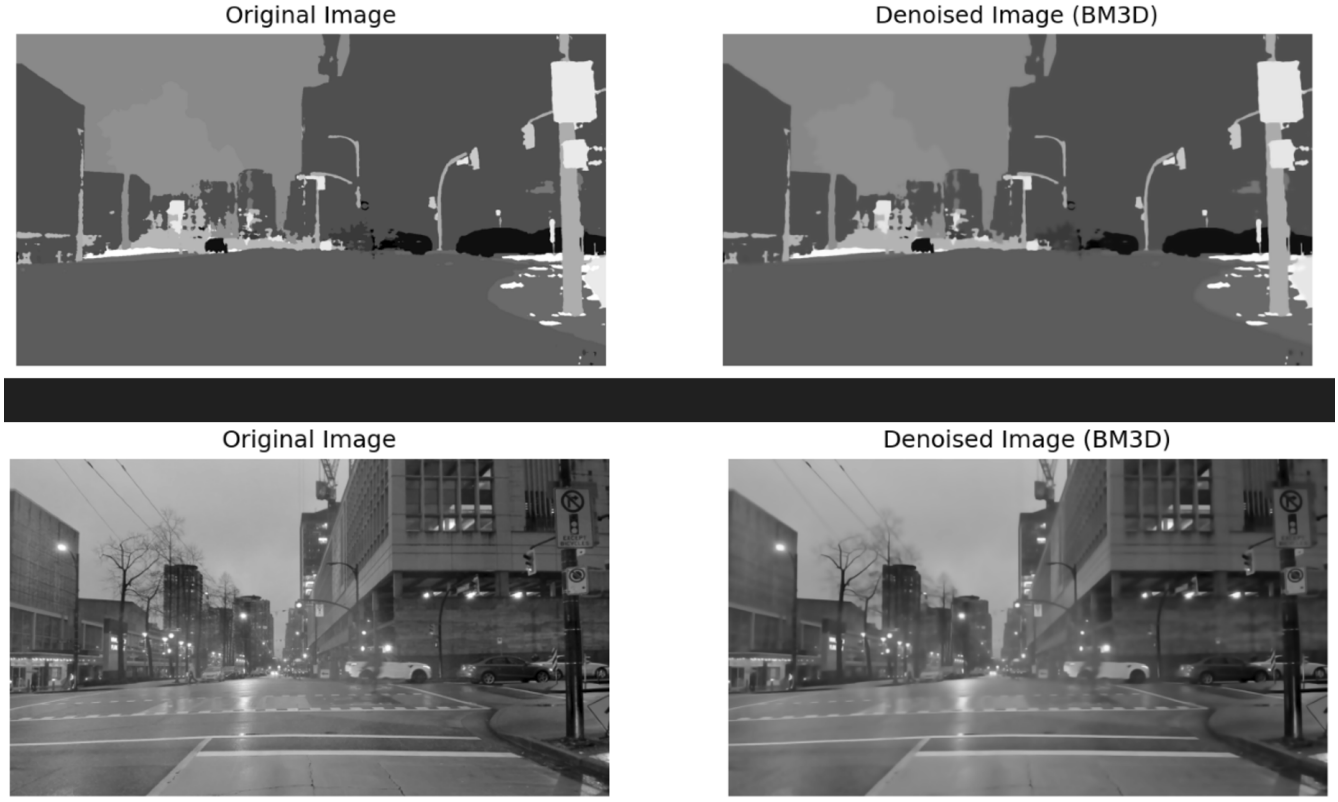

Figure 5: BM3D results with sigma =0.2

The result of both BM3D and Image Prior were not satisfactory therefore we have selected our first method.

1 Results

1.1 Training and Validation Curves

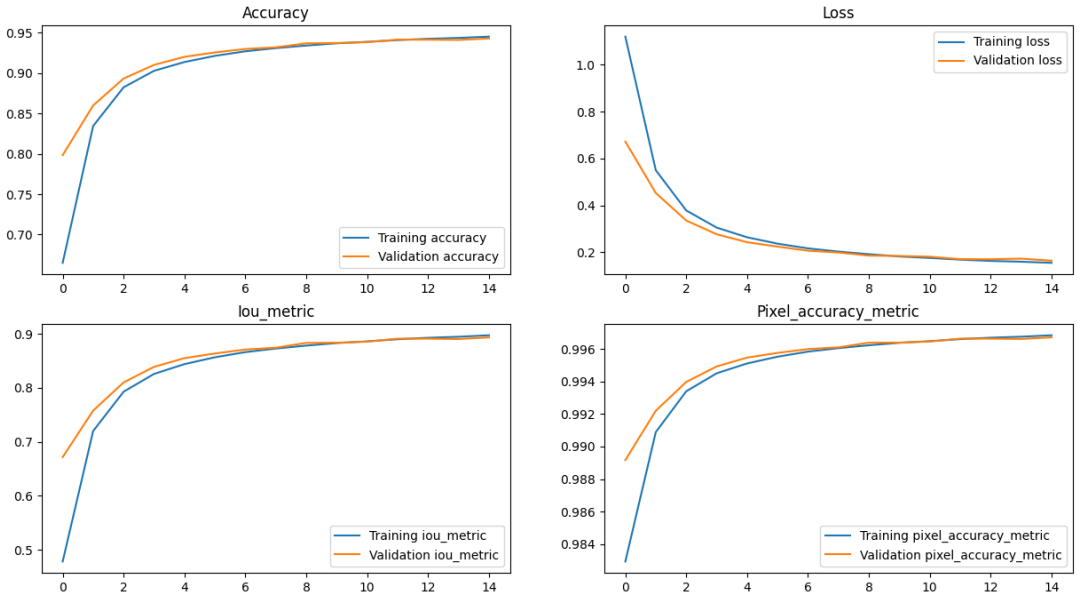

Figure 6: Model Performance - Accuracy, IoU, and Loss Curves

The training accuracy increases steadily, but the validation accuracy lags slightly, indicating some overfitting. Similarly, while both training and validation IoU curves rise, the validation curve is less smooth and plateaus at a lower value. The loss curves for both training and validation show a decrease, but the validation loss starts to increase after a few epochs, further indicating overfitting. Pixel accuracy follows a similar trend, with training performance slightly outpacing validation.

1.2 Accuracy vs Mean IoU

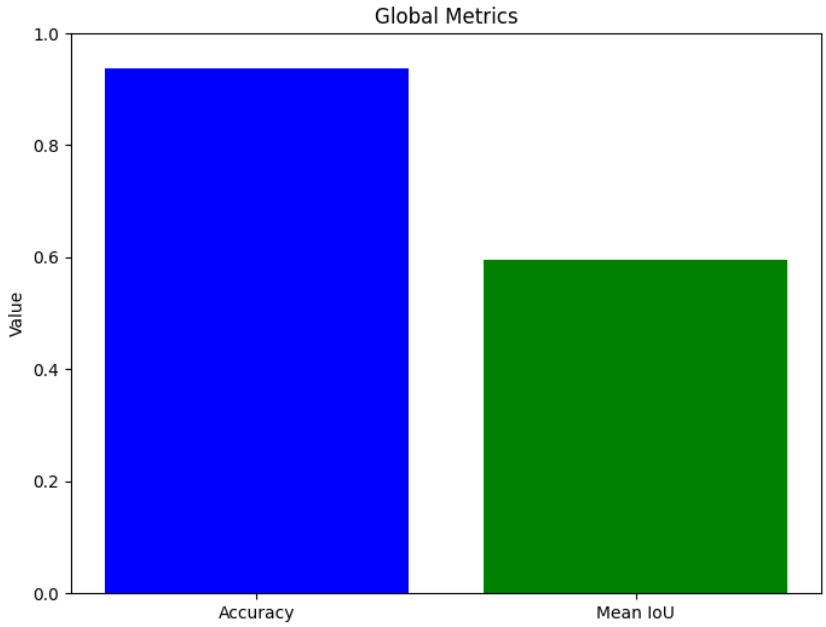

Figure 7: sunny Global Metrics and Performance Evaluation

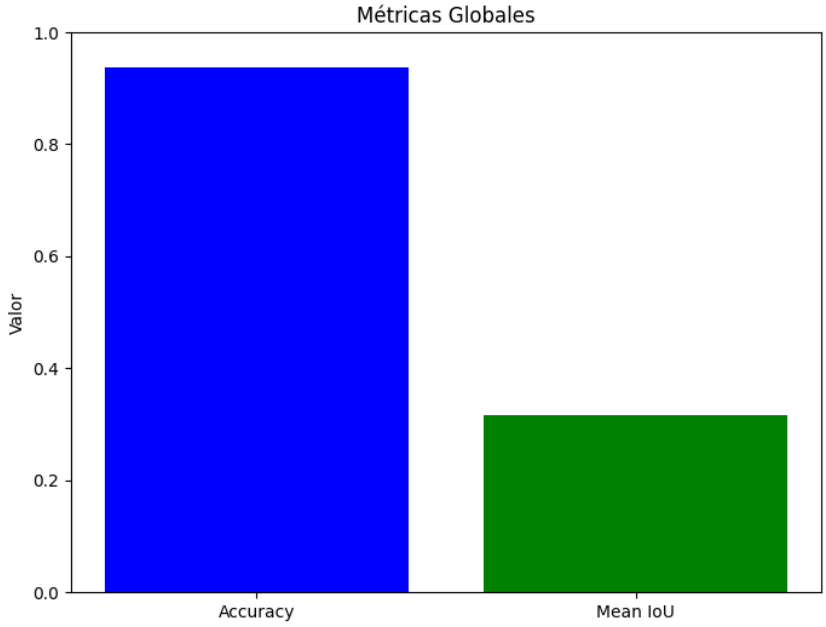

Figure 8: Rainy Global Metrics and Performance Evaluation

The model shows a high overall accuracy, close to 1, indicating good performance in pixel classification. However, the Mean Intersection over Union (IoU) is significantly lower, suggesting that while the model correctly classifies most pixels, it struggles with the

quality of the segmentation, particularly in delineating class boundaries. This discrepancy points to challenges in handling finer details, especially in the presence of class imbalances.

2 Outptus

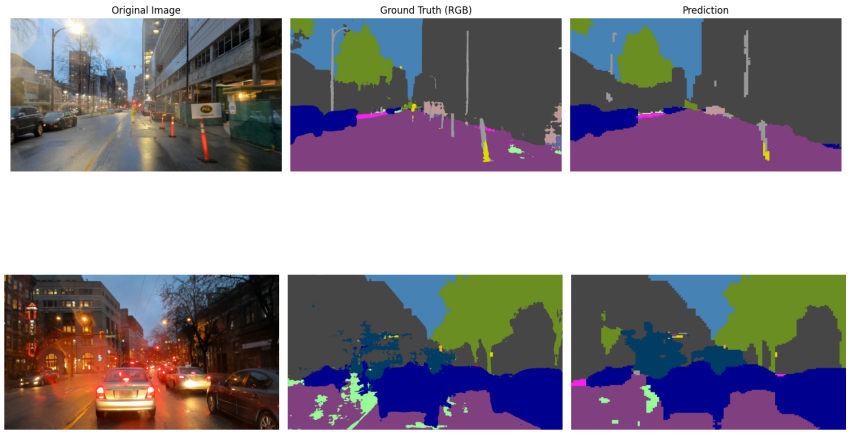

Figure 9: Model Results for Rainy

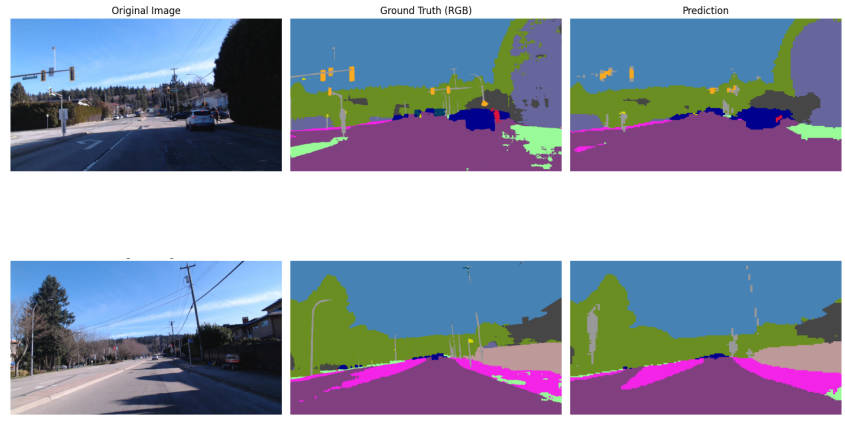

Figure 10: Model Results for Rainy

3 References

References

[1] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *arXiv preprint arXiv:1711.10925*, 2017.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv preprint arXiv:1505.04597*, 2015.