Summer 2021

Methods for Detecting Floodwater on Roadways from Ground Level Images

Cem Sazara
Old Dominion University, csaza001@odu.edu

METHODS FOR DETECTING FLOODWATER ON ROADWAYS FROM GROUND LEVEL IMAGES

by

Cem Sazara
B.S. June 2011, Bogazici University, Turkey

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTATIONAL MODELING AND SIMULATION ENGINEERING

OLD DOMINION UNIVERSITY
August 2021

Approved by:

Mecit Cetin (Director)

Khan M. Iftekharuddin (Member)

Michel A. Audette (Member)

Zhanping Liu (Member)

ABSTRACT

METHODS FOR DETECTING FLOODWATER ON ROADWAYS
FROM GROUND LEVEL IMAGES

Cem Sazara
Old Dominion University, 2021
Director: Dr. Mecit Cetin

Recent research and statistics show that the frequency of flooding in the world has been increasing and impacting flood-prone communities severely. This natural disaster causes significant damages to human life and properties, inundates roads, overwhelms drainage systems, and disrupts essential services and economic activities. The focus of this dissertation is to use machine learning methods to automatically detect floodwater in images from ground level in support of the frequently impacted communities. The ground level images can be retrieved from multiple sources, including the ones that are taken by mobile phone cameras as communities record the state of their flooded streets. The model developed in this research processes these images in multiple levels. The first detection model investigates the presence of flood in images by developing and comparing image classifiers with various feature extractors. Local Binary Patterns (LBP), Histogram of Oriented Gradients (HOG), and pre-trained convolutional neural networks are used as feature extractors. Then, decision trees, logistic regression, and K-Nearest Neighbors (K-NN) models are trained and tested for making predictions on floodwater presence in the image. Once the model detects flood in an image, it moves to the second layer to detect the presence of floodwater at a pixel level in each image. This pixel-level identification is achieved by semantic segmentation by using a super-pixel based prediction method and Fully Convolutional Neural Networks (FCNs). First, SLIC super-

pixel method is used to create the super-pixels, then the same types of classifiers as the initial classification method are trained to predict the class of each super-pixel. Later, the FCN is trained end-to-end without any additional classifiers. Once these processes are done, images are segmented into regions of floodwater at pixel level. In both of the classification and semantic segmentation tasks, deep learning-based methods showed the best results. Once the model receives the confirmation of flood detection in image and pixel layers, it moves to the final task of finding the floodwater depth in images. This third and final layer of the model is critical as it can help officials deduce the severity of the flood at a given area. In order to detect the depth of the water and the severity of the flooding, the model processes the cars on streets that are in water and calculates the percentage of tires that are under water. This calculation is achieved with a mixture of deep learning and classical computer vision techniques. There are four main processes in this task: (i)-*Semantic segmentation of the image into pixels that belong to background, floodwater, and wheels of vehicles*. The segmentation is done by multiple FCN models that are trained with various base models. (ii)-*Object detection models for detecting tires*. The tires are identified by a You Only Look Once (YOLO) object detector. (iii)-*Improvement of initial segmentation results*. A U-Net like semantic segmentation network is proposed. It uses the tire patches from the object detector and the corresponding initial segmentation results, and it learns to fix the errors of the initial segmentation results. (iv)-*Calculation of water depth as a ratio of the tire wheel under the water*. This final task uses the improved segmentation results to identify the ellipses that correspond to the wheel parts of vehicles and utilizes two approaches listed below as part of a hybrid method: (i)-*Using the improved segmentation results as they return the pixels belonging to the wheels.* Boundaries of the wheels are found from this and used. (ii)-*Finding arcs that belong to elliptical objects by*

applying a series of image processing methods. This method connects the arcs found to build larger structures such as two-piece (half ellipse), three-piece or four-piece (full) ellipses. Once the ellipse boundary is calculated using both methods, the ratio of the ellipse under floodwater can be calculated. This novel multi-model system allows us to attribute potential prediction errors to the different parts of the model such as semantic segmentation of the image or the calculation of the elliptical boundary. To verify the applicability of the proposed methods and to train the models, extensive hand-labeled datasets were created as part of this dissertation. The initial images were collected from the web, then the datasets were enriched by images created from virtual environments, simulations of neighborhoods under flood, using the Unity software.

In conclusion, the proposed methods in this dissertation, as validated on the labeled datasets, can successfully classify images as a flood scene, semantically segment the regions of flood, and predict the depth of water to indicate severity.

ACKNOWLEDGMENTS

I would like to thank everyone who helped in the development of the research that led to this dissertation. I express my deepest appreciation to my supervisor, Dr. Mecit Cetin, for his continuous support and supervision throughout this process. Additionally, I would like to extend my thanks to Dr. Khan M. Iftekharuddin for his support and help in the development of this research work and my committee members Drs. Audette and Liu for their valuable feedback.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

We provide the background information, contributions of this dissertation and the outline of this work in this chapter.

1.1 Background

Recent studies indicate that flooding frequency has risen worldwide. Heavy rain, storm surge and sea-level rise have especially impacted flood-prone communities. Sweet and Park (2014) predicted that, by 2050, recurrent flooding due to sea level rise may occur for thirty of more days in US cities. Floods can cause significant damages to human life, damage properties, inundate roads, overwhelm drainage systems, and disrupt essential services and economic activities. Flash floods due to heavy rain in short amounts of time cause major damages to communities. A flash flood fatalities analysis by Terti et al. (2017) reported 63,176 flash flood events in the US including Alaska, Hawaii, and Puerto Rico between 1996 and 2014. Given the critical nature of floods, it is important to monitor and analyze them. A major impact of floods is on the transportation system. It is important to provide safe routing of emergency vehicles in addition to safe traffic management during floods. Many critical operations depend on a functioning transportation system. The significance of the problem has motivated many researchers to develop flood detection and warning systems. A detailed literature review on these works is provided in the next chapter. At high level, there are three main areas of interest: measuring water levels with contact and non-contact type gauges, flood inundation maps and digital elevation models, and image processing and computer vision-based methods. The methods proposed in this research work belong to the last type: image processing and computer vision-based methods. A short summary of these is as follows:

1-Measuring water levels with contact and non-contact type gauges: Contact level gauges are in direct contact with water. They can be under the water (pressure type), over the water (floating type), or having parts both over and under the water (staff gauges and rulers). These are usually low-cost sensors. Non-contact water sensors are not in direct contact with water. Some examples include radar, ultrasonic sensors, or imagery.

2-Flood inundation maps and digital elevation models: Flood inundation maps and digital elevation models (DEM) have been used to measure the extend of water with data collected from satellite images or High Resolution (HR) Synthetic Aperture Radar (SAR). Differences between the flood and non-flood time measurements show the extend of flood.

3-Image processing and computer vision-based methods: With the availability of image data and recent progress in machine learning research, researchers worked on developing algorithms to solve flood related problems. This is also the direction followed in thesis. Detection is usually done in three ways: 1-Detecting the presence of flood in the image. This is simply image classification (Lopez-Fuentes et al. 2017; Sazara et al. 2019). 2-Detecting the extend of flood in images. This is usually done with semantic segmentation that gives pixel-level information (Witherow et al. 2018; Witherow et al. 2019; Sazara et al. 2019; Sarp et al. 2020). 3-Predicting floodwater depth from images. This problem deals with predicting floodwater depth using familiar objects in the images. Meng et al. (2019) and Chaudhary et al. (2019) used end-to-end trainable object detectors to predict water depth of the detected objects while Geetha et al. (2017) used some fixed shaped boxes for the parts of the objects. Although the end-to-end systems by Meng et al. (2019) and Chaudhary et al. (2019) are easy to train, their prediction errors are difficult to explain. Park et al. (2021) recently proposed a method with additional processes after detecting vehicles. They matched

the detected 2D vehicle patches with 2D image projections of 3D vehicle models. At the end, the water depth was calculated from the projected image. Our proposed method in this dissertation is similar to Park et al.'s (2021) method in its nature. Our method is a multi-model system that not only predicts floodwater depth, but also semantically segment the image. Additionally, we proposed a water depth metric that is proportional to the wheel size in vehicles. The water depth can be calculated in real size once the sizes of the tire wheels are known, but this is beyond the scope of this work. Next, contributions of this research are summarized.

1.2 Contributions

This dissertation proposes a collection of image processing and machine learning methods to solve multiple problems with flood-water detection. As mentioned earlier, there are three main problems to solve: 1-Detecting the presence of flood in the image with image classification 2-Semantically segmenting images (in pixel level) into flood and non-flood areas. 3-Predicting floodwater depth in images. The proposed methods here address all three of these problems. We worked with real flood images collected from web search and some synthetic images created using the Unity software (Unity 2021). Lack of labeled data, especially for the second and third problem makes this work challenging. Using a program to create the synthetic images also allowed us to create their labels automatically. The real images were manually labeled. The images used in this work are from ground level (not top view such as satellite imagery) and can be assumed to be taken from the height of a regular person. To solve the first two problems, multiple image classifiers and semantic segmentation methods were proposed. For the last problem (water depth prediction), we proposed a multi-model system. We used vehicles as

objects of interest and calculated the water depth using the tires of the vehicles. At the end, the water depth was calculated as a ratio of tire wheel under the water. This is a novel method. At high level, these are the steps of this method:

- **Step 1:** An image is segmented into flood and non-flood regions using Fully Convolutional Neural Network.
- **Step 2:** A YOLO object detector returns the tires in vehicles.
- **Step 3:** A custom U-Net like semantic segmentation network is used to improve the initial segmentation results from Step 1. Improvement is only done on the tire locations from the step 2.
- **Step 4:** In the last step, we solve an ellipse fit problem. Ellipse is fit on the wheel part of the tire using the information from step 2 and step 3. The water depth is the ratio of this ellipse under the water.

In the following, we list the main contributions of this work:

- Create a hand-labeled public dataset of flood images from multiple sources from internet. One of the difficulties of this project is the lack of publicly available datasets. To solve this problem, we collected flood images from web and labeled them. They were labeled for all problems: Classification, semantic segmentation and water depth prediction.
- Generate a synthetic dataset that is made of flood images with vehicles from a virtual environment created with the Unity software (Unity 2021). In connection with the first contribution, the synthetic data creation allowed us to get images that were automatically labeled. Training sets used a mix of real and synthetic images. The test and validation sets did not have any synthetic images to make sure results were calculated for real images and models were tuned with the real images. This helped us build a labeled

dataset in a quick way. Additionally, considering the relevant literature, this is the first research work that used synthetic images to find flood in images in these problems: Classification (image level), segmentation (pixel level) and floodwater depth prediction.

- Comparison of classical feature extraction methods and deep network features for classification and segmentation of flood images.
- A new method to solve the pixel-level segmentation of flood in images and prediction of water-depth at the same time. Compared to the other methods in literature, this method has multiple explainable components that make use of deep learning and image processing methods.
- A new metric to measure water depth in images. This metric is the ratio of the wheel of a tire under the water. With this metric, we do not need to know the real dimensions of vehicle tires and can produce relative predictions.
- A semantic segmentation refinement method that works on selected regions and improves the initial semantic segmentation results.

1.3 Structure of the Dissertation

The remainder of this dissertation is structured like this: Chapter 2 reviews the related work and explains the main areas of research interest. Chapter 3 explains and compares multiple methods for image classification and semantic segmentation of flood images. Chapter 4 discusses a method to semantically segment and calculate water depth in images with flood. Chapter 5 addresses the conclusions and discussions.

CHAPTER 2

RELATED WORK

In this section, we go over the relevant work in the literature. Due to the growing number of floods and their impacts on people, infrastructures, and economy, there have been increased interest in methods to estimate and measure floods. Overall, we can group these works into three main categories below. We review each category in this chapter.

- **Contact and non-contact type gauges:** These methods use gauges (contact and non-contact types) to measure water level. Depending on the method, they may also require image processing.
- **Flood inundation maps and digital elevation models:** Methods in this group use top-view data such as satellite images and High Resolution (HR) Synthetic Aperture Radar (SAR) readings.
- **Image processing and computer vision-based methods:** This group heavily relies on processing of images from flood events. Images are usually collected from social media or crowdsourcing.

2.1 Contact and non-contact type gauges

These methods use sensors to measure water levels. In general, they can be contact or noncontact gauges (Nair and Rao 2016). Contact level gauges are in direct contact with water. They can be under the water (pressure type), over the water (floating type), or having parts both over and under the water (staff gauges and rulers). These are usually low-cost sensors. Non-contact water sensors are not in direct contact with water. Some examples include radar,

ultrasonic sensors, or imagery. These are usually more expensive than contact level sensors. Both the contact and non-contact sensors usually cover fixed locations to verify their measurements (Nair and Rao 2016). Staff gauges are usually considered as both under and above the water. In these systems, water-level can be measured by recognizing the characters on staff gauges. Some image processing methods such as image thresholding (Sun et al. 2013; Sun et al. 2014; Lin et al. 2013) and edge detection (Iwahashi and Udomsiri 2007; Shin et al. 2008, Young et al. 2015) are used to read the measurements. Due to their fixed locations and pre-selected parameters, these systems are limited to their installed locations.

2.2 Flood inundation maps and digital elevation models:

Flood inundation maps and digital elevation models (DEM) have been used to measure the extend of water with data collected from satellite images or High Resolution (HR) Synthetic Aperture Radar (SAR). Some of these systems are used as flood warning systems (Hawker et al. 2018; Konadu and Fosu 2007; Neussner et al. 2012). These systems usually compare the measurements with baseline data and calculate differences. The demand for flood inundation mapping services have increased due to increased frequency flooding (Cohen et al. 2018). The Dartmouth Flood Observatory (DFO) (Kettner and Brakenridge 2021) and the Copernicus Emergency Management Service (CEMS) provide almost real-time flood inundation maps from satellite imagery. DFO provides Moderate Resolution Imaging Spectroradiometer (MODIS) imagery and high-resolution imagery for certain flood areas. CEMS provides on-demand high-resolution images. United States Flood Inundation Mapping Repository (USFIMR) (Cohen 2021) provides sensor-based flood maps from previous flood events in the US.

2.3 Image processing and computer vision-based methods:

With the recent advancements in machine learning, there have been increased interest in the machine learning-based methods to solve flood prediction problems. Most of these systems require labeled datasets to learn from. Data collected from publicly available images and videos have become important data sources in this case. The method proposed in this dissertation belongs to this group. We can group this type into three main categories in terms of the problems we are solving:

i) Classifying the image as involving flood or not

ii) Semantic segmentation (pixel-wise classification) of the flood area in the image

iii) Floodwater depth prediction in the image

In the classification group, Lopez-Fuentes et al. (2017) used a multi-model deep learning approach to classify flood images. Their model used an ensemble of CNN and LSTM to process image and text metadata. The model proposed by Sazara et al. (2019) addressed both the classification and semantic segmentation problems. They trained multiple classifier and semantic segmentation networks.

Some papers worked solely on the semantic segmentation problem. Sarp et al. (2020) used a Mask R-CNN model to segment the areas corresponding to flood-water. Witherow et al. (2018) compared images of scenes with and without flood and found pixels belonging to flood area. Their method used image registration followed by intensity thresholding. In another work, Witherow et al. (2019) added image inpainting to their list of methods. Both of their methods depended on predetermined thresholds for filtering, registration and pixel intensity thresholds.

For flood-water depth prediction, most of the papers worked with object detectors. They detected certain objects in the floodwater and predicted the water depth from them. The most frequently used objects were humans and vehicles submerged in the water. Geetha et al. (2017) used an object detector to detect people submerged in the water. Then, they fit fixed-shaped boxes around body parts starting with the head and continuing with the rest of the body. Meng et al. (2019) and Chaudhary et al. (2019) used end-to-end trainable Mask R-CNN detectors for water depth prediction. Meng et al. (2019) focused on detecting humans while Chaudhary et al. (2019) trained their model for five categories: person, bus, car, house, and bike. These models output the water depths for the detected objects. Feng et al. (2020) also focused on using humans as the objects of interest in finding the water depth. Differently from the two previous methods, they found the key points that belong to body parts and calculated the water depths from them. Kharazi and Behzadan (2021) studied submerged stop signs from US and Canada. They used Hough transform to calculate the length of the pole under the water. Park et al. (2021) proposed another method to predict flood water depth. In their method, they first used the Mask R-CNN to detect vehicles. Then, they matched the detected 2D vehicle patches with 2D image projections of 3D vehicle models. The water depth at the end was calculated using parts of the tires under the water. Errors were calculated in two ways: The pixel error and ratio of pixel error to fixed size of tire. Although this method gave more intuition to water depth prediction compared to the black-box methods by Meng et al. (2019) and Chaudhary et al. (2019), it had some limitations. It had to find a suitable 3D model for each vehicle. Additionally, the vehicle tires needed to be fully visible.

2.3.1 Deep Learning:

Deep learning allows us to build deep neural networks to learn representations and solve difficult problems. Recently, deep learning models achieved state of the art results in different areas such as natural language processing, computer vision, recommendation systems, and reinforcement learning. Specifically, in computer vision, there have been significant developments and superb results in image classification (Krizhevsky et al. 2012; Simonyan and Zisserman 2014; He et al. 2016; Howard et al. 2017), object detection and tracking (Wang and Yeung 2013; Girshick 2015; Liu et al. 2016; Redmon and Farhadi 2018) semantic segmentation (Shelhamer et al. 2015; Ronneberger et al. 2016; Chen et al. 2017) and instance segmentation (He et al. 2017; Gao et al. 2019).

Deep neural networks are made of multiple layers. Figure 1 shows an example network with two hidden layers. Inputs are shown with x_1, x_2, ..., x_n where n is the number of inputs. There are multiple units in each hidden layer: Denoted with h_{11}, h_{12}, ..., h_{1k} where k is the number of units in the first layer and h_{21}, h_{22}, ..., h_{2m} where m is the number of units in the second layer. Output is shown with o_1.

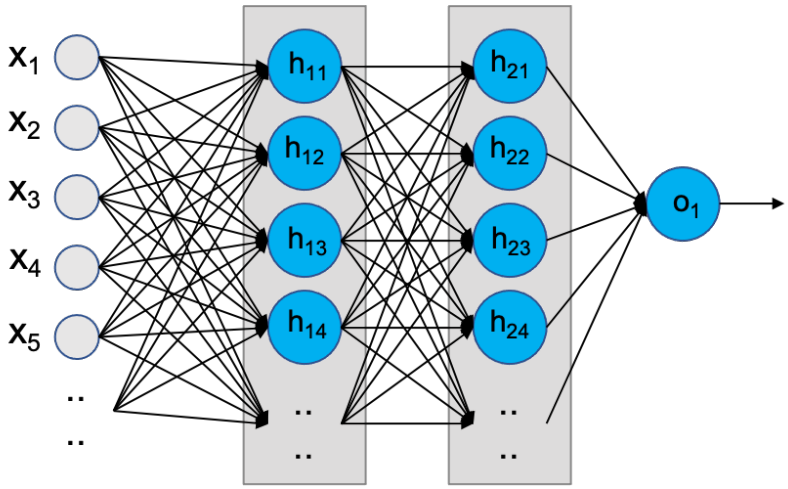

Figure 1. Neural Network

These networks are trained using mathematical optimization methods with the backpropagation algorithm. With this algorithm, the weights in the network are updated iteratively and the network starts learning and improving after each update. Stacking multiple layers allows the models to learn hierarchical features. Activation functions are applied after each layer. Different activation functions result in different activations. Rectified linear unit (ReLu) has been a popular activation. This function is efficient and useful when building deep networks thanks to its simple equation and its constant first order derivative which becomes critical during the training of deep neural networks (Nair and Hinton 2010). Its equation is shown in Equation 1.

$$ReLu(x) = max(0, x) \quad (1)$$

A single artificial neuron with its activation function is depicted in Figure 2. Inputs x_1, x_2, ..., x_n are multiplied with the corresponding weights and summed up. Then, an activation function **"f"** is applied on the summation. The output of the neuron becomes the output of the activation function.

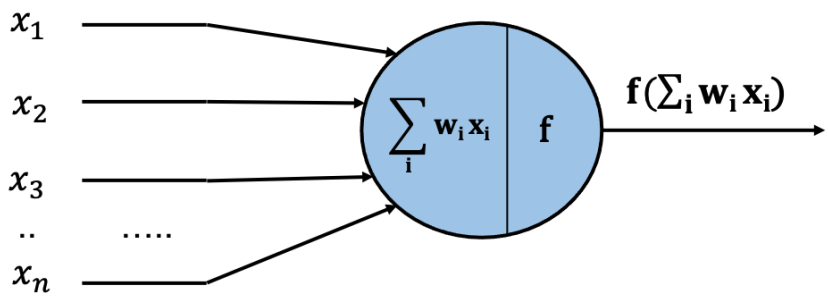

Figure 2. Artificial neuron

Gradient descent is one of the most used training algorithms when training neural networks. It is an iterative optimization method. In this method, a cost function is selected and optimized with a given training set. During optimization, multiple repeated steps are taken in the opposite direction of gradient. Gradient is calculated as the first order derivative of the cost function with respect to each parameter or weight in the network. The weights are updated using the gradients and selected learning rate to scale the gradient value. The formula is shown in Equation 2. Learning rate is selected using a hold-out validation set or cross-validation.

$$w_{updated} = w_{old} - learning\_rate * gradient\_w \quad (2)$$

Usually, a minibatch approach is used in the training. With this approach, gradients are calculated for each batch of the training data. Then, the average batch gradient is used to update weights. Optimum selection of the batch size is also selected with a validation set or cross-validation. Learning rate and batch size are considered hyper-parameters that need to be selected. Table 1 shows the cost functions that are minimized in the training process. In these equations, n is the number of data points, y is the label, p is the output of the network and ln is natural logarithm.

Table 1. Common cost functions

Problem	Cost Function	Equation
Binary classification	Cross-entropy loss	$-\frac{1}{n}\sum_{\text{examples}} y \ln p + (1-y)\ln(1-p)$
Multiclass classification	Cross-entropy for softmax	$-\frac{1}{n}\sum_{\text{examples}}\sum_{\text{classes}} y_j \ln p_j$
Regression	Mean Squared Error	$\frac{1}{n}\sum_{\text{examples}} (y-p)^2$

2.3.2 Convolutional Neural Network (CNN):

CNNs are one of the most common networks in deep learning. CNNs have multiple layers like regular neural networks with the addition of operations such as convolution and pooling. Convolution is a well-studied mathematical operation that have been used beyond convolutional neural networks for a long time. A simple CNN is made of three types of layers: Convolution, pooling and fully connected layer. Convolution and fully connected layers have learnable weights. Weights in the convolution layers are kernels. These kernels learn important representations from input data. Figure 3 shows a 2D convolution example and Figure 4 is the result after the convolution result goes through the ReLU activation function.

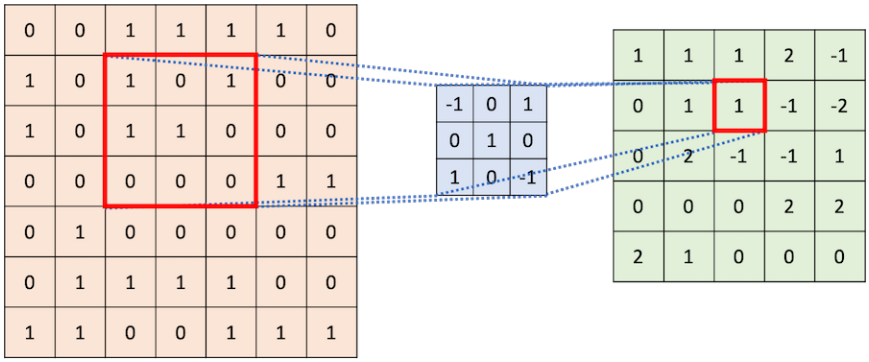

Figure 3. 2D Convolution

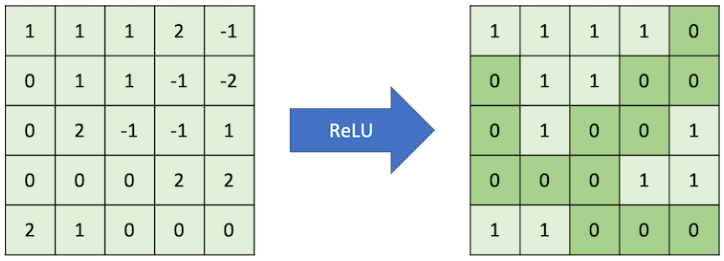

Figure 4. ReLU activation function

Pooling operation (layer) is used to reduce the height and width of the feature maps. This operation is applied with a pre-defined window size and stride. Average and maximum pooling operations are shown in Figure 5. Fully connected layers are usually used at the end of CNNs. They are regular neural network layers that can be built as multi-layers. A simple example is shown in Figure 6. In this example network, there are 4 convolutional layers, 2 pooling layers and 2 dense layers in total.

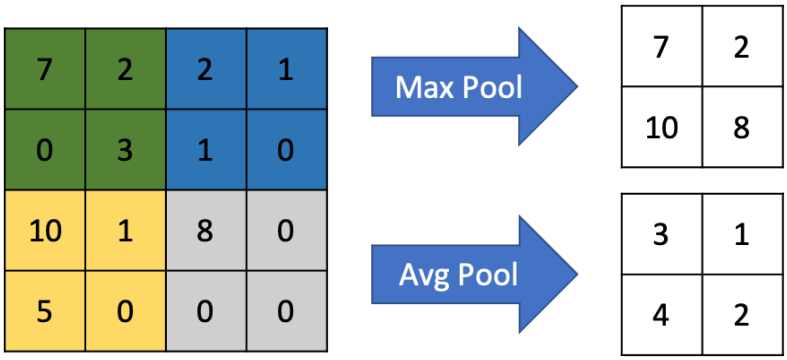

Figure 5. Pooling operation

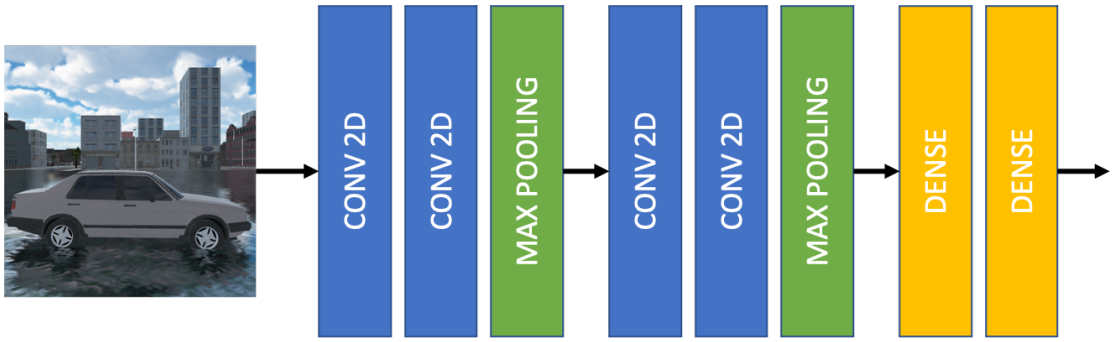

Figure 6. Example CNN

This dissertation uses some of the explained concepts from this section and develop some new ones. We train multiple Convolutional Neural Networks and develop new methods to extract detailed information from images such as pixels belonging to floodwater and floodwater depth. Chapter 3 and Chapter 4 explain these methods in detail.

CHAPTER 3

IMAGE CLASSIFICATION AND SEMANTIC SEGMENTATION OF FLOOD SCENES

In this section, we discuss multiple methods for two main tasks: (i) Classify given input images into two categories: Involving flood or not (ii) Segment images into regions of flood. In computer vision, feature is some useful information in images. This useful information can be thought as some structures such as edges, texture, color or objects that we can use to solve a problem. In general, features are usually different depending on the problem or area that we are working on. Feature extraction in computer vision is usually the process of finding useful information in images. We worked with different methods for feature extraction and making the predictions in these tasks. We start with image classification. For feature extraction, Local Binary Patterns (LBP), Histogram of Oriented Gradients (HOG) and pre-trained convolutional neural were used. Decision Tree, Logistic Regression and K-Nearest Neighbors models were trained and tested for making predictions. For the semantic segmentation task, we worked with super-pixel based prediction method and Fully Convolutional Neural Networks (FCNs). We used the SLIC superpixel method to create superpixels, then used classifiers to train and predict the class for them. FCN method is an end-to-end trainable model. Therefore, we trained it with the image-label pairs. Table 2 shows a summary of the used methods for the problems in this section. Overall, there are 9 combinations of feature extractors and classifiers.

Table 2. Methods for classification

Feature Extraction	Classifiers
Local Binary Pattern (LBP)	Decision Tree, Logistic Reg., KNN
Histogram of Oriented Gradients (HOG)	Decision Tree, Logistic Reg., KNN
Pre-trained CNN: VGG-16	Decision Tree, Logistic Reg., KNN

For semantic segmentation, there are overall four combinations: Decision Tree, Logistic Regression and K Nearest Neighbors for the superpixel based method and the FCN method (Table 3).

Table 3. Methods for semantic segmentation

Feature Extraction	Superpixel Classifiers
Superpixel and hand-crafted features	Decision Tree, Logistic Reg., KNN
Fully Convolutional Neural Network (FCN)	-

We used a dataset of 491 images with fixed shape of 384x512. 253 of them had flood in them and the remaining 238 didn't. Comparison of the proposed methods and the results are later explained in this chapter.

3.1 Feature Extractors:

Feature extractors are used to construct features from raw input data in machine learning. In the context of computer vision, with the recent advances in neural networks, feature extraction with deep neural networks has become a common practice. In this section, a comparison between three different feature extractors is presented. We cover and compare Local Binary Patterns (LBP), Histogram of Oriented Gradients (HOG) and deep learning (VGG-16 model).

3.1.1 Local Binary Patterns (LBP):

Local Binary Patterns (LBP) method is a feature extractor for images. It was introduced by Ojala et al. (1994). It is a simple and effective method to understand patterns in images. LBP is a visual descriptor that compares pixels with their neighbors in a given region and creates binary codes. Figure 7 shows an example for this. In this example, LBP is applied on a 3x3 window and a binary code is generated for the pixel at the center.

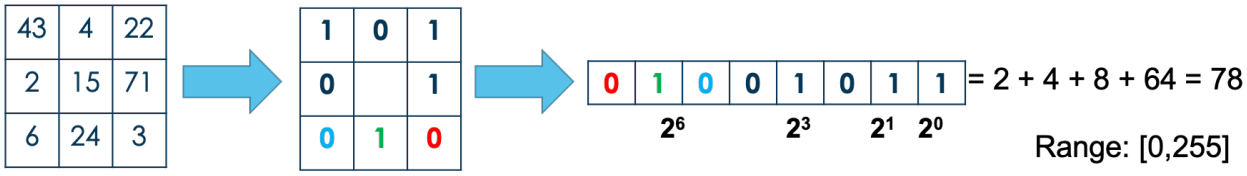

Figure 7. Local Binary Patterns

Each pixel is compared with its neighbors following a clockwise or anti-clockwise circle. If the pixel value is greater than a neighboring pixel value, we place 1 (and 0 otherwise). This gives a 4-digit binary number for 4 connected neighbors (top, down, right and left) and 8-digit binary number for 8 connected neighbors (top, top-left, top-right, down, down-left, down-right, right, left). These binary numbers are converted to decimals. In order to understand the patterns in a region of an image, these decimal values are accumulated in histograms for blocks such as 16x16, 32x32 etc. At the end, we get a list of histograms for each image. Once concatenated, this produces a single list of values for each image and this list becomes the extracted features. In our case, 16x16 blocks with 10-bin histograms gave (384/16)*(512/16)*10 =7,680 features for each 384x512 image.

3.1.2 Histogram of Oriented Gradients (HOG):

Histogram of Oriented Gradients (HOG) method is another feature extractor for images. This was a popular method mainly used for object detection (Dalal and Triggs, 2005). As stated in its name, this method uses histograms for the image gradients and constructs features from these histograms. There are two main steps:

- We compute gradients in x, y, and diagonal directions (total 8) within blocks of an image (16x16) and construct gradient histograms.
- Histograms are normalized to values between 0-1 and concatenated.

In our case, 16x16 blocks with 8-bin histograms gave (384/16)*(512/16)*8 =6,144 features for each 384x512 image. See Figure 8 below with a sample image and its HOG feature descriptor.

Figure 8. Sample image and its HOG feature descriptor

3.1.3 Feature Extraction with Deep Learning:

With the advances in deep learning, convolutional neural networks achieved state-of-the art results in image related tasks such image classification, object detection, semantic segmentation and instance segmentation. Transfer learning has been an important factor in the success of convolutional neural networks. Transfer learning enables the use of a previously trained network to perform another task. This process sometimes involves re-training of the pre-trained network (fine-tuning) or in other cases, the pre-trained network can be used as a fixed network to only make forward pass through it. In this section, the second approach was used. A pre-trained VGG16 network (Simonyan and Zisserman, 2014) was used as feature extractor. Then, multiple predictors were trained with the extracted features. Figure 9 shows the VGG-16 architecture and the 5th pooling layer that the features were extracted from. This architecture has 13 convolution and 5 poling layers in total. Convolutions and pooling operations are placed as blocks. At the end of the model, there are 3 dense layers. These dense layers were not used in our problem.

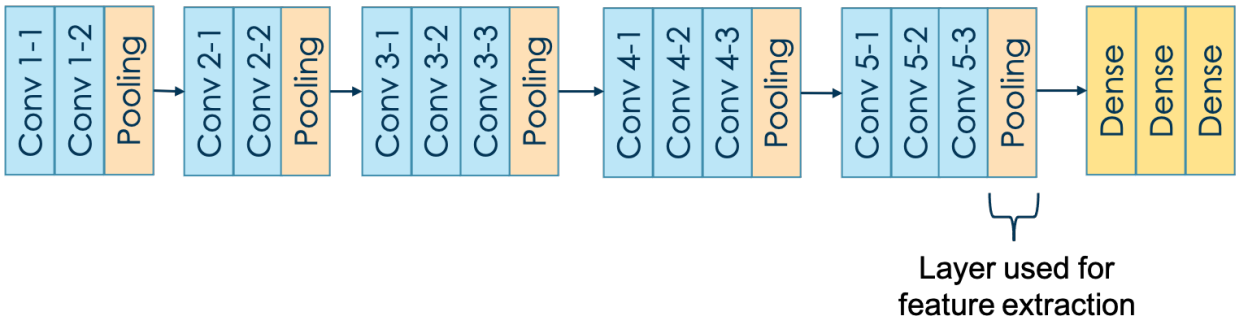

Figure 9. VGG-16 model and the layer used for feature extraction

For input images with shape (384x512x3), features at the 5th pooling layer has the following shape: (12, 16, 512). To use these features with other predictors, we flattened them and got

features with the shape (98304, 1) at the end. This network was pre-trained on the "ImageNet" dataset. We only used it to create these features for each image. Extracted features were saved and later used to train different classifiers.

3.2 Classifiers:

Classification is a main-stream machine learning task where a machine learning model is trained to predict discrete categories that correspond to classes. In this section, we used the features from Local Binary Patterns (LBP), Histogram of Oriented Gradients (HOG) and VGG-16 deep neural network. Three different classifiers were used with these features.

3.2.1 Logistic Regression:

Logistic regression is a classification method. In this method, input variables are multiplied by coefficients and mapped to discrete classes using a sigmoid function.

$$\hat{y} = sigmoid(\beta_n x_n + \cdots + \beta_2 x_2 + \beta_1 x_1 + \beta_0) \quad (1)$$

where $x_1 \ldots x_n$ are inputs, $\hat{y}$ is output, $\beta_0 \ldots \beta_n$ are the coefficients

The sigmoid function is shown in Figure 10.

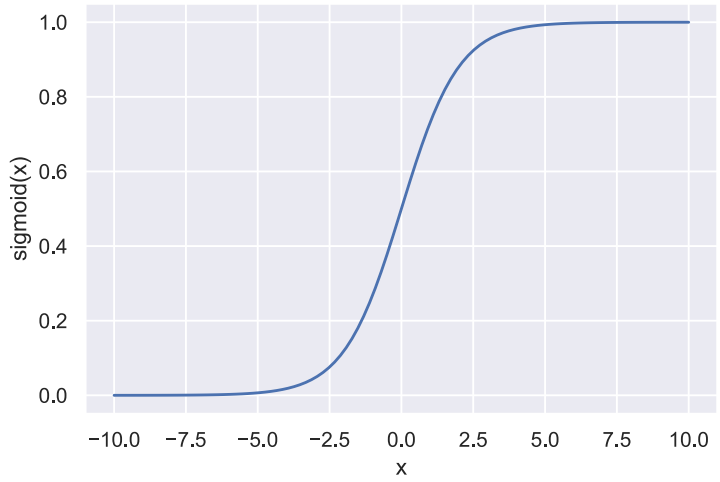

Figure 10. Sigmoid function

The parameters in Equation 1 are learned from datasets with different mathematical optimization methods. In our case, these parameters are learned from the extracted features.

3.2.2 K-Nearest Neighbors (KNN) Method:

K-Nearest Neighbors model is a simple predictor that uses K closest data points. It can be used for regression and classification. For regression, the output becomes the average of the K data points. For classification, the output is the majority vote from the K nearest data points. “K” is a positive number and needs be determined beforehand (hyperparameter). If K is selected too small, the model can include noise in the data and may not perform well. If K is too large, the model doesn’t reflect the general properties of the training data. KNN is considered an instance-based learning method. To find the K closest data points, this method calculates the distances between the data points. A well-known distance metric in this algorithm is the Minkowsky metric. It is shown in Equation 2.

$$dist\_Minkowsky(A,B) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r} \quad (2)$$

where r is the power parameter

Hyperparameters are selected using a validation set or with cross-validation. A visual demonstration of this model with K=3 to predict the class of a datapoint (shown with question mark) is in Figure 11 below.

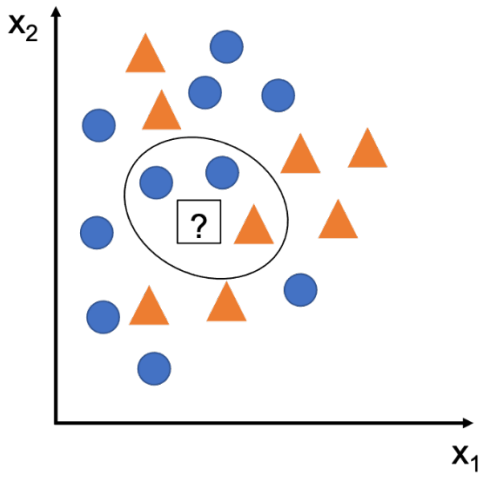

Figure 11. KNN with K=3 to predict the class of a data point

3.2.3 Decision Trees:

Decision Tree (DT) is a predictor that works like a flowchart. The flowchart-like structure is built with rules similar to if-else statements. To build a tree, one can simply use the impurity concept and grow the tree looking for child nodes with the lowest average impurities. A simple decision tree is shown in Figure 12. In split nodes, questions using the features from the dataset are asked and data points are split into two groups. Leaf nodes return the outputs.

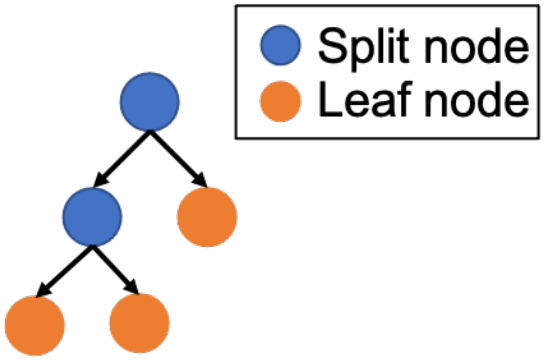

Figure 12. A simple Decision Tree

Gini impurity measure is a common metric when building trees. This metric can be calculated using the distribution of given data points. For classification, it has its maximum value with the highest impurity (50%-50% for two classes) and its minimum with the lowest impurity when one class is the 100% of the data samples (Equation 3).

$$impurity(p_1, \ \ldots, \ p_k) = \sum_{k=1}^{n} p_k(1 - p_k) \tag{3}$$

where k: class index, n: number of classes and p_k: is the probability of class k

As decision trees are built in a "greedy" way using the best choice at the moment, it is easy to overfit on training datasets. Pre-pruning is a common regularization method with trees. With this technique, certain constraints can be placed on trees such max depth of the tree, max number of splits allowed, max number of leaf nodes and min impurity decrease for each split. These hyperparameters can be selected using a validation set or cross-validation.

3.3 Semantic Segmentation:

Semantic segmentation problem deals with classifying pixels in images. It can be considered as recognizing what is in the image in pixel level. An example segmented image is shown in Figure 13. In this research work, two main approaches were investigated. First, the

super-pixel based segmentation method was used. Superpixel is a group of pixels that share common characteristics such as similar intensity and being close to each other position-wise. With this type of segmentation, all pixels within the same superpixel group gets assigned the same class. The overall assignment for each group is learned in a supervised way. The second approach in this problem is to use convolutional neural networks. The Fully Convolutional Neural Networks (FCNs) proposed by Shelhamer et al. (2015) are effective network architectures that can be trained end-to-end with raw image and the segmented label image pairs.

Figure 13. Segmented floodwater image

3.3.1 Superpixel based segmentation:

Superpixels were generated using the Simple Linear Iterative Clustering (SLIC) method. This method partitions the image into non-overlapping regions. Each region is considered a cluster of pixels with similar intensity values and proximities in the image plane. This is done using a set of five values: [l, a, b, x, y] where l, a and b correspond to the color in CIELAB color space and x,y is the position for each pixel. A distance measure is used to find and group similar pixels. The distance equation is given like as in Equation 4. The distance for the color values (l,

a, b) and position (x, y) are weighted and there are two weight parameters m and S. S is used to specify the search area 2Sx2S to find similar pixels. Figure 14 shows an image and its SLIC superpixels.

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$

$$D_s = d_{lab} + \frac{m}{S} d_{xy} \quad (4)$$

Figure 14. SLIC example: (left) Original image (right) Superpixels

In this problem, each superpixel was assigned a class (flood or not-flood) based on the majority class of the pixels within that superpixel. A feature vector was constructed for each superpixel. As shown in Equation 5, center locations x and y, average intensities of R, G and B channels and LBP feature histogram. Logistic regression and KNN models were trained with superpixel vectors and their labels as training samples.

$$v_i = [center_x, center_y, avgR, avgG, avgB, LBP\_histogram] \quad (5)$$

3.3.2 Fully Convolutional Neural Networks (FCNs):

Semantic segmentation is a computer vision problem where we predict categories of pixels in the given input images. With the recent success in deep learning, most of the recent methods utilize deep learning and convolutional neural networks. In this work, Fully Convolutional Neural Networks (FCNs) were used. Fully Conv. Neural Network was first proposed by Shelmar et al. (2015) and it quickly became a popular architecture for semantic segmentation task. FCN architecture uses a Conv. Neural Network as backbone network. In the next steps, output from the backbone network gets upsampled and the class prediction for each pixel is made at the output layer. Upsampling is necessary as the height and width of feature maps gets smaller after a series of convolution and pooling layers in the base network. See Figure 15 below for details.

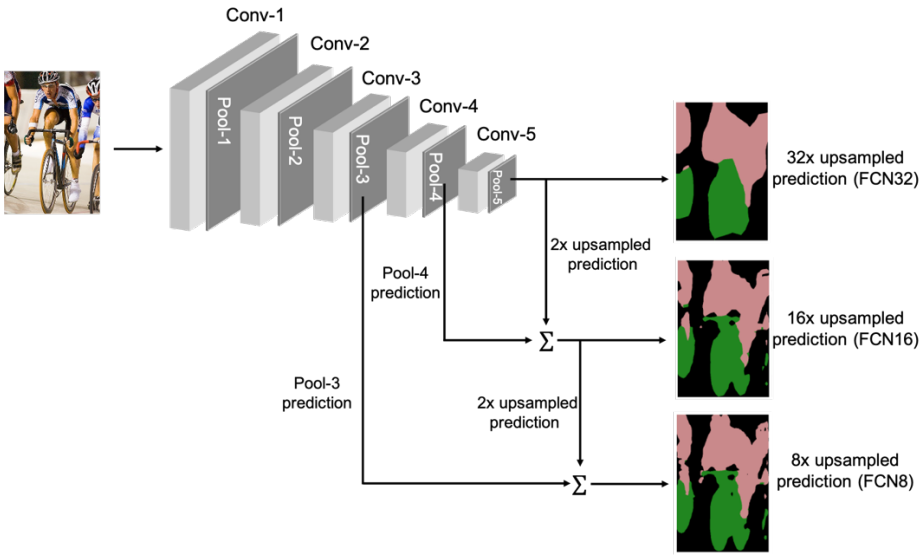

Figure 15. Fully Convolutional Neural Network

Overall, there are three main types of FCNs: FCN8, FCN16 and FCN32. The numbers in the name of FCN model: 8, 16 and 32 specify the upsampling rates. FCN32 model uses the pool 5 layer's output and upsamples it 32 times. FCN16 similarly upsamples the output from the pool 4

layer and adds-in the 2 times upsampled pool 5 result. At the end, the summation is upsampled 16 times. FCN8 uses the same idea adding the Pool 3 result and upsampling the summation 8 times. FCN8 and FCN16 utilize feature maps with different resolutions. Upsampling can be done in two ways: Bilinear upsampling and transposed convolution. Transposed convolution allows the model to learn the weights to do the upsampling.

Conditional Random Fields (CRFs) are usually used to smooth the semantic segmentation results. In this case, a graph structure is implemented to model the pixels and their connections. For this, using G as a graph over (x, y) data points where x is a vector [x_1, x_2, ..., x_v] where v is the total number of pixels in the image and y is the label vector for the pixels. y can get values from {y_1, y_2, ..., y_k} where k is the number of classes. Every pixel x has a corresponding label y with probability P(y/x) and it is modeled with Hidden Markov Model where the conditional probability is only dependent on the current position and its adjacent pixels (i.e. Markov property). CRF minimizes a two-part energy function shown in Equation 6.

$$E(X) = \sum_{i \epsilon V} \psi(x_i) + \sum_{i,j \epsilon V} \psi(x_i, x_j) \tag{6}$$

where $\psi(x_i)$ is the unary term, $\psi(x_i, x_j)$ is the pairwise term and x_i and x_j are adjacent pixel values. Unary term is the pixel class probability and the pairwise term smoothens adjacent pixels. Overall, the labels that result in the smallest energy value E(X) are used.

3.4 Dataset:

Our dataset is made of 491 images. 253 of them has floodwater in it and 238 images don't. For image classification task, these images are used as two classes. For semantic segmentation, only the images with flood are used. There are also two classes flood or not-flood

for them this time in pixel level. The flood image dataset contains different scenes from urban, suburban and natural settings and they hand-labeled. Pixels corresponding to flood areas have value of one and the rest of the pixels are zero.

3.5 Implementation:

For the classification task, we used 253 flood images and 238 other images without flood. The dataset was split into 80% training and 20% test sets. As it was shown earlier in Table 1., we used three feature extractors: LBP, HOG and pre-trained VGG-16. After the features were extracted, we used three different classifiers to train and test with the corresponding datasets. The classifiers are Logistic Regression, K-Nearest Neighbors method and Decision Tree. This gives a total of 9 combinations of feature extractors and classifiers. We used grid-search with 5-fold cross-validation to find the optimum hyperparameters of these models. Table 4, Table 5 and Table 6 show the hyper-parameters for these models.

Table 4. Hyperparameters for Logistic Regression models - classification

Feature Extractor	Regularization coefficient	Regularization type	Class balanced
HOG	1.5	L2	No
LBP	75	L2	Yes
VGG-16	0.1	L2	Yes

Table 5. Hyperparameters for KNNs - classification

Feature Extractor	K
HOG	6
LBP	20
VGG-16	7

Table 6. Hyperparameters for Decision Trees - classification

Feature Extractor	Max Depth	Max Features	Leaf Min Samples	Class balanced
HOG	20	78	4	No
LBP	20	88	10	Yes
VGG-16	20	98304	10	Yes

For semantic segmentation, we used only the images with flood. That gave us 253 images in total. An 80%-20% split was used for training and test datasets. Overall, the following two methods were trained and tested: Fully Convolutional Neural Network (FCN) and superpixel based segmentation. For the FCN approach, the FCN-8 implementation was used with a pre-trained VGG-16 base network. VGG-16 was previously trained on the ImageNet dataset. We trained the FCN model with our training dataset for 18 epochs. We used the Stochastic Gradient Descent algorithm for training with batch size of 4 and learning rate of 0.001. Additionally, 10% of the training data was split before the training to use as validation set. At the end, the model was tested with the test images. For the superpixel based method, we used the SLIC method with 250 regions to produce the superpixels. Hyperparameters for the models for the superpixel based approach are the following:

- K-NN model: {K=24}
- Logistic Regression model: {regularization_coefficient=2.5, class_balanced=No, regularization_type: L2}
- Decision Tree: {max _depth=3, max_features=10, min_samples_leaf=10}

3.6 Results:

In this section, we summarize our results. Results are given separately for the classification and semantic segmentation problems. Classification results are presented in Table 7.

The VGG-16 based method with the Logistic Regression classifier achieved the highest scores in precision and F1 scores. Although the KNN with VGG-16 base had a higher score in recall, it did worse in the other metrics. Overall, for all feature extraction methods, logistic regression model achieved the highest scores in F1 metric.

Table 7. Classification Results

Feature extraction	Classifier	Precision	Recall	F1-score
LBP	Logistic Regression	0.72	0.83	0.77
	KNN	0.58	0.79	0.67
	Decision Tree	0.54	0.62	0.57
HOG	Logistic Regression	0.78	0.85	0.82
	KNN	0.57	0.94	0.71
	Decision Tree	0.65	0.6	0.62
VGG-16	Logistic Regression	**0.95**	0.89	**0.92**
	KNN	0.62	**0.96**	0.75
	Decision Tree	0.8	0.77	0.78

Similarly, we present the semantic segmentation results here in Table 8. The main comparison is between the Superpixel + hand-crafted features method and Fully Convolutional Network (FCN) method. This time, the results seemed closer to each other compared with the classification results. The FCN method achieved the highest scores in all metrics: Precision, recall and f1. Superpixel-based logistic regression and K-Nearest Neighbors methods followed it.

Table 8. Semantic Segmentation Results

Feature extraction	Classifier	Precision	Recall	F1-score
Superpixel and hand-crafted features	Logistic Regression	0.90	0.89	0.89
	KNN	0.83	0.89	0.86
	Decision Tree	0.87	0.88	0.88
Fully Convolutional Neural Network (FCN)		**0.92**	**0.90**	**0.91**

Some example results for semantic segmentation are shown below in Figure 16.

Figure 16. Some FCN segmentation results

3.7 Conclusion:

In this research work, we used different approaches to classify and segment flood images. In addition to using different models, we also released a public dataset to contribute to the developments in this area. We presented a comparison between "classical" approaches and deep learning-based methods. In both of the problems, the deep learning-based methods resulted in better performance in F1 scores. The results for the semantic segmentation problem were closer to each other compared to the classification problem. For the semantic segmentation problem, use of more advanced architectures can potentially increase the scores. As future work, we can

experiment with more advanced base networks and segmentation methods. Additionally, water reflection became a problem in some instances for the semantic segmentation problem. Some pixels corresponding to water reflections were misclassified as not water. Further research into water reflection detection and removal can help improve the results further.

CHAPTER 4

FLOOD-WATER DEPTH PREDICTION ON ROADWAYS FROM SIDE-VIEW IMAGES

In this section, a novel method to predict water depth in flood images is proposed. Figure 17 shows the set of methods at a high level. As input, side-view images of vehicles were used. These images can be easily collected through an internet search, crowdsourcing or from roadside surveillance cameras. To find the water depth, we used vehicle tires as reference objects. Instead of using the real-world dimensions of the tires, we came up with a metric that measured the percentage of the vehicle tire under the water. Doing this has some advantages: (i) The proposed method doesn't require any camera calibration to predict the real-world dimensions of the objects in the images (ii) Model validation can be done with images collected in a similar way from the internet (iii) Vehicle tire or wheel sizes are usually standard for a given vehicle make and model. If the vehicle make and model could be accurately predicted, the ratio that we calculated here could be converted into real-world dimensions with a reasonable accuracy. However, predicting the vehicle class, make and model is outside of the scope of this research work. Some recent studies showed promising results in vehicle make and model prediction from images with high accuracy: 97.89% (Manzoor et al. 2019), 96.33% (Lee et al. 2019) and 98.43% (Jamil et al. 2020). Additionally, vehicles are assumed to be traveling at low speeds or not in motion as high speeds create waves and splashes around the tires which makes finding the water depth difficult. This restriction may be limiting the overall applicability of the proposed method, but this system can be deployed at locations where are low or vehicles can be stopped such as signalized or stopped controlled intersections and local roads with parked cars. The overall

flowchart of the processes is shown in Figure 17. We use a single image containing a vehicle and produce water depth prediction at the end. At high level, these are the main processes:

- **Semantic Segmentation:** The input image gets semantically segmented into three classes: the wheel part of the tire, water, and background. We used the wheel parts of the tires because they are easier to segment compared to the full tire. Dark colors of tires make it difficult and most of the time tire boundaries get lost in the dark parts of the fenders of vehicles. We used a Fully Convolutional Neural Network (FCN) (Shelhamer et al. 2015) with different pre-trained base networks for this task.
- **Object Detection:** A YOLO (Ultralytics 2020) object detector was used in this section. This detector was trained to detect tires in images. The locations of the detected tires and the semantic segmentation result go to the next process.
- **Semantic Segmentation Refinement:** As the tires are usually small objects in the images, the initial semantic segmentation result can yield some coarse segmentations. To improve the initial segmentation, a semantic segmentation refinement network was proposed here to improve the initial segmentation results.
- **Water Depth Calculation:** A novel method to calculate water depth in flood scenes is proposed in this section. The method uses the refined segmentation result from earlier and uses some image processing and mathematical methods to find ellipses in the tire regions of vehicles. These ellipses are targeted for the wheel parts of the tires.

In the remainder of this section, we review these four main processes and explain each in greater detail and explains them in greater detail.

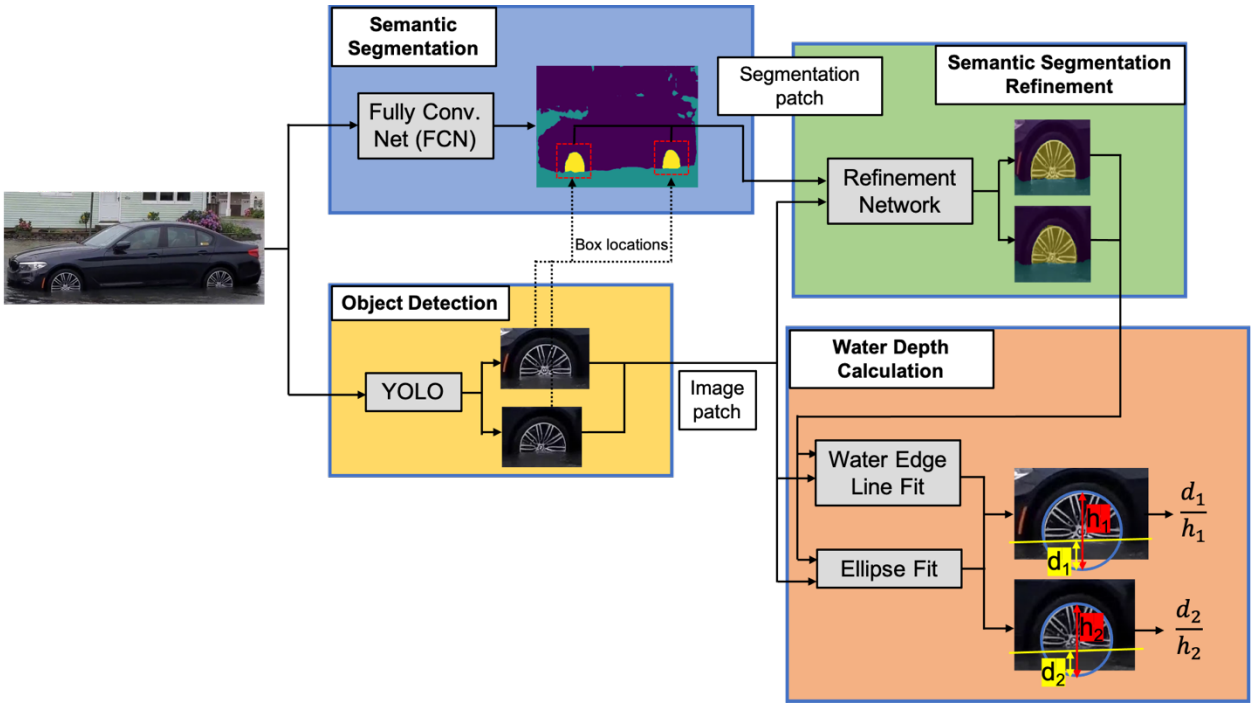

Figure 17. Flowchart of the proposed method

4.1 Semantic Segmentation

In this project, we used the FCN8 model as it combines a wider range of feature resolutions compared to FCN16 and FCN32. We used three different base networks: VGGNet by Simonyan and Zisserman (2014), MobileNet by Howard et al. (2017) and ResNet by He et al. (2016) and trained the model to classify pixels for three categories: Background, tire wheel, and water. Figure 18 shows an example image and the segmented image afterwards. We used the tire wheels instead of outer boundaries of tires. In our experiments, we realized that tire boundaries (of the tire rubber) are difficult to distinguish due to residing in the dark regions in vehicles. Contrast between the wheels and the tire rubber allows our models to learn to segment the wheels much better. The resolution of the input images was set to 384x512. As the wheels are

usually small objects in the images, results from the semantic segmentation networks can sometimes yield coarse results in these areas. For this reason, we also trained another set of

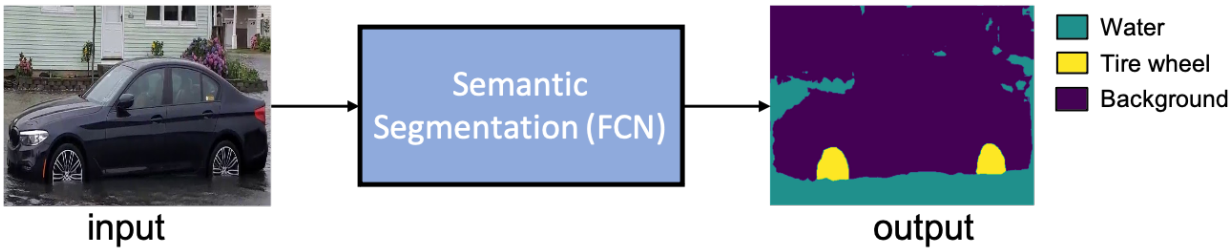

Figure 18. Semantic segmentation example

semantic segmentation networks that specialize in improving the initial semantic segmentation results. The improvement networks use the initial semantic segmentation result and the raw image corresponding to the segmented areas as inputs. They are trained so that their outcomes are better than the initial segmentation results. Details of these networks are later described in this chapter. As the improvement network needs the raw image as one of its inputs, we need to provide a raw image patch corresponding to the tire. We used a YOLO object detector for this.

4.2 Object Detection

We used an object detector to locate tires of vehicles in this project. Object detection problem deals with finding the instances of objects in images. In this problem, object detectors produce locations and classes of objects in images. Locations are given as bounding boxes. These are the boxes that encapsulates the objects within the images. Each box also has a class for the corresponding object. The recent success of convolutional neural networks also resulted in significant progress in the object detection problem. The following are some notable object detectors.

In this project, YOLO v5 (Ultralytics 2020) is used. One of the biggest advantages of YOLO architecture is its speed and simplicity. Compared to the other mentioned object detectors, YOLO trains and makes predictions faster due to its simple architecture overall. YOLO object detectors allow us to make almost real-time object detections. Additionally, they are also end-to-end trainable systems with given label and image pairs. The initial YOLO model (Redmon and Farhadi 2018) has gone through a few improvements. We used the YOLO v5 here to locate vehicle tires. To train this object detector, we used our flood images and some additional vehicle images from the CompCar dataset by Yang et al. (2015). We also applied some custom data augmentation on the CompCar images by cutting the lowers parts of the vehicles at random locations, thereby creating non-complete tires to mimic the flood situations where the full tires are not visible most of the time. Some examples are shown below in Figure 19.

Figure 19. Example images from the CompCar Dataset (Yang et al. 2015) with data augmentation applied

4.3 Semantic Segmentation Refinement

Tires are usually small objects in the images. The semantic segmentation results from the earlier segmentation network can result in some coarse results. We propose a refinement method here to improve the initial segmentation results. This is an important task, because errors in the semantic segmentation results will be passed to the next steps and eventually lead to incorrect water depth predictions. To fix the initial segmentation results, we used a simplified U-Net architecture. U-Net model by Ronneberger et al. (2015) is a semantic segmentation method

similar to fully convolutional neural networks. Unlike the FCNs, the U-Net architecture uses higher resolution feature maps. These feature maps are connected to multiple convolution layers and concatenated with other upsampled feature maps afterwards. Our implementation is a lighter version of the original U-Net with smaller number of layers and convolutions. Additionally, our implementation uses two inputs instead of one. The inputs are the raw image patch corresponding to the tire (coming from the YOLO detector) and the initial semantic segmentation result corresponding to the same location as the raw tire patch. The network learns to produce improved semantic segmentations for the input images. Figure 20 below shows the details of this architecture. Overall, the architecture resembles a “U shape”. The raw tire image and its initial segmentation result are resized to 128x128 shape and concatenated. Then, this concatenated input goes through multiple convolution-pooling operations. On the right side, there are convolutions and upsampling operations. There are also some connections between the left and right part of the architecture (shown with gray color in Figure 20). These connections allow this model to utilize the high-resolution feature maps. Overall, there are 21 layers including the upsampling and pooling layers with smaller number of convolutions than the original implementation. In this task, having a light network worked because this task learns to improve instead of learning from scratch. The improvements coming from this network is later reported in the Results section.

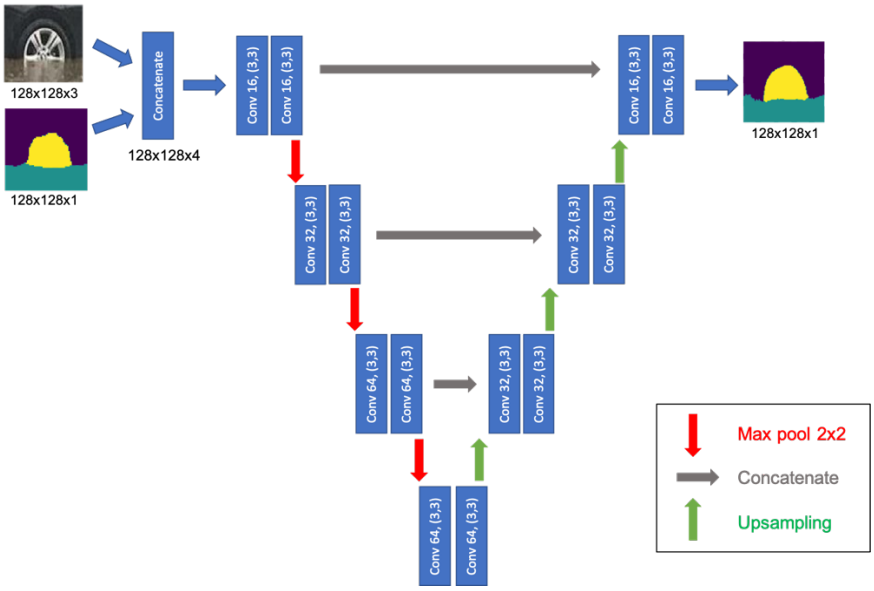

Figure 20. The proposed U-Net like architecture

4.4 Water Depth Calculation

After the semantic segmentation improvement, the next operation is to calculate the water depth. There are three main processes here: Fitting a line to the water edge, finding the ellipse equation for the tire wheel and calculating the water depth as a ratio of tire wheel under the water.

4.4.1 Water Line Fit

It is important to locate the where the water resides in the tire image. For this, a simple approach is to fit a line to the water edge. Any part that is under this line is considered water. In this process, the input is the refined semantic segmentation result. Then, the water edge pixels are found by extracting the border between the tire wheel and water from the segmentation result. At the end, a simple linear regression fit to the border pixels gives the line equation. Figure 21 shows this process. The next step deals with fitting ellipses to the tire wheels.

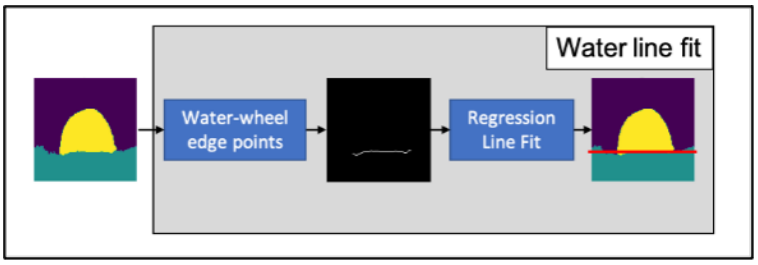

Figure 21. Water edge line fit process

4.4.2 Ellipse Fitting Method

In this task, we try to find an ellipse equation for each wheel of the vehicle. An ellipse has these parameters: Ellipse radius values a, b in x and y directions, center point (x_c, y_c) and tilt angle ∅. A simple ellipse with these parameters is shown below (Figure 22).

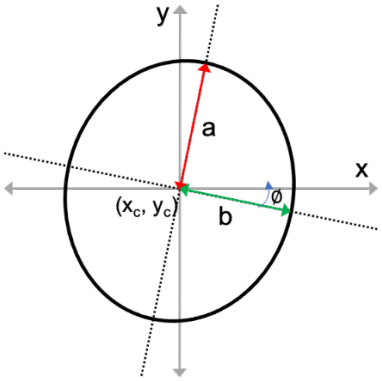

Figure 22. Ellipse with parameters

Similarly, an ellipse equation can be written using the conic equation with an additional constraint as shown in equations 7 and 8.

$$F(x,y) = ax^2 + bxy + cy^2 + dx + ey + f = 0 \quad (7)$$

with ellipse constraint

$$b^2 - 4ac > 0 \quad (8)$$

The coefficients in Equation 7 are learned from the data points. Data points are extracted from the wheel parts of the tires. Before explaining the details of this process, we mention a few challenges:

i) **Number of data points:** As this research deals with images with flood, tires are most of the time partially submerged in the water. This gives data points for partial ellipses, but the equation needs to be calculated for the full ellipses.

ii) **Misclassified pixels:** Although there is a semantic segmentation refinement step, there will still be misclassified pixels. These errors can impact the next step of fitting a geometric shape to the data points.

iii) **Other similar shapes nearby:** It is likely to have other objects in the areas of interest for this problem. For example, fenders or the outer boundaries of tires can be easily mistaken for the wheels.

The ellipse fit process is summarized in Figure 23 below. There are two inputs in this process: Semantic segmentation result and the corresponding tire image. We applied multiple processes on these inputs afterwards.

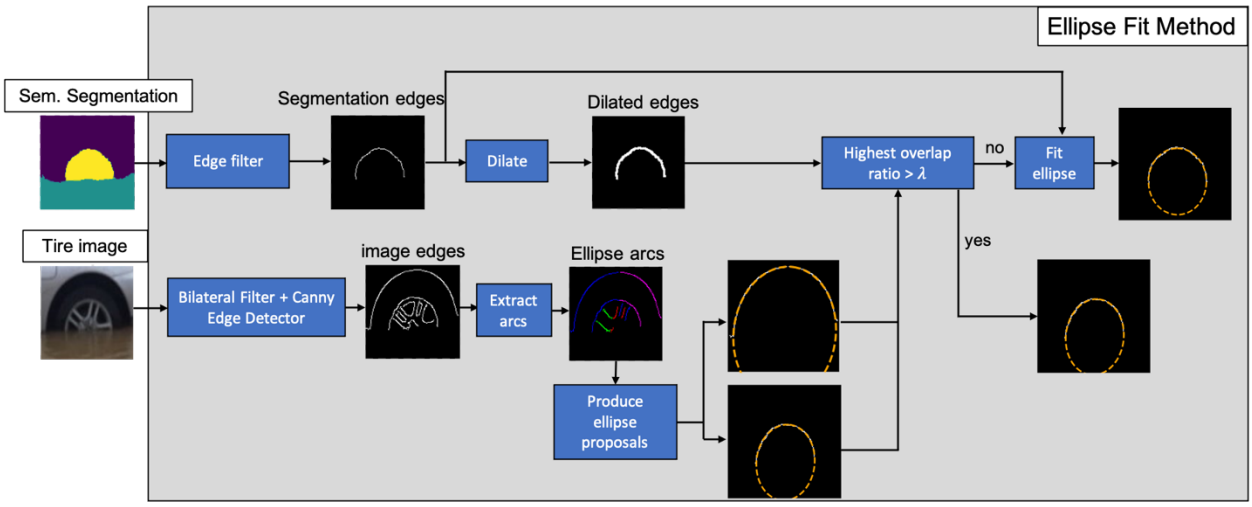

Figure 23. Ellipse fit method

Starting with the raw tire image, this image goes through bilateral filter and canny edge detector. The bilateral filter preserves the edges and smooths the image at the same time. In the next step, an arc-based ellipse proposal method is used. This method constructs arc pieces in four main regions of an ellipse: Top-left, top-right, bottom-left and bottom-right. Details of this method are explained later. These arc pieces are later merged to create 2-piece (half), 3-piece or 4-piece (full) ellipses. Then, we compare this ellipse with ellipse coming from the refined semantic segmentation image.

Following the path that starts with the refined semantic segmentation result, we start with finding the edges that correspond to the tire wheel. Then, we dilate this result and compare it with the ellipse from the previous arc-based ellipse proposal method. If the overlap is larger than a pre-defined threshold, we accept the arc-based proposal ellipse as the end result. If the overlap is small, this time we use the semantic segmentation edges. This two-way approach helps use get high quality ellipses. At the end, we fit an ellipse on the data points using the least squares method and find the coefficients of it. See Equation 7 for the coefficients. We ran our experiments using both of the methods (hybrid) and only the upper part that used the refined segmentation result. We report the results of those in the results section.

- **Extracting and Merging the Arc Pieces:**

It is important to convert edges into meaningful shapes. This is the main goal of this section. Here, we proposed a method to extract edges, build arcs from them and merge these arcs to get bigger structures like 2-piece (half), 3-piece or 4-piece (full) ellipses at the end. This

section corresponds to the “Extract arcs” and “Produce ellipse proposals” blocks in Figure 23. The proposed method in this research is similar to another ellipse construction method by Fornaciari et al. (2014). The method by Fornaciari et al. (2014) and its different forms have been used for solving different problems such as bike tire detection (Eldesokey et al. 2017), robotic manipulation of objects (Dong et al. 2018) and powerline equipment inspection (Siddiqui et al. 2018). Our problem of fitting ellipses into tire wheels under flood conditions is a special case because, most of the time, we deal with incomplete ellipses. This frequently happens as the vehicle tires submerge in the floodwater. To solve this problem, we introduced a hierarchy rule. Fornaciari et al. (2014) used a simple clock-wise rule (top-right, bottom-right, bottom-left, top-left) to merge arc pieces into bigger structures. This does not work in our problem. In our case, the bottom parts of the wheels may be submerged in the water. Here, we imposed a hierarchy constraint that forces to merge the top arc pieces (top-left and top-right) first.

To start with, we grouped edges into four main regions of an ellipse. After that, we applied our hierarchy rule and started merging them. Grayscale image gradients and convexity information were used for the region assignment. Figure 24 shows the gradients, convexities and the resulting region assignments. In that figure, we can see the conditions for each region assignment such as negative gradient and positive convexity giving region 1, positive gradient and positive convexity giving region 4, etc.

Gradient signs were calculated using the horizontal and vertical gradients: G_x and G_y. Each pixel has a value for G_x and G_y. Then, using these gradients, a sign was calculated for each

pixel. The equation for the sign is shown below. The overall sign was calculated by multiplying the signs of G_x and G_y as shown in Equation 9.

$$sign(\tan \varphi(p)) = sign\big(G_x(p)\big).sign(G_y(p)) \tag{9}$$

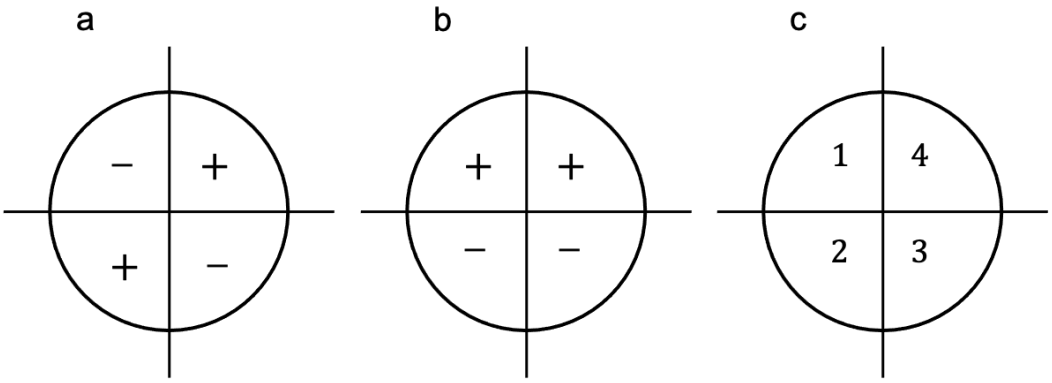

Figure 24. Region assignment: a-) Gradient signs b-) Convexity signs c-) Final regions

In Figure 24 a-), top-left and bottom-right correspond to negative signs and top-right and bottom left are positive signs. After the gradient sign assignment, we connected the pixels using 8 directional connectivity. This gave positive and negative arc sets: A_{pos} and A_{neg} respectively.

Next, the signs for convexity were calculated for the positive and negative arc sets. This part corresponds to the Figure 24 b-). Positive sign there means a convex set and the negative sign means concave. Arc convexity-concavity was calculated by comparing the mid-point of each arc with the mid-point of a straight line that went through the end points of that arc. If the midpoint of the line was lower than the midpoint of the arc, it was considered as a convex arc, otherwise, it was concave. Figure 25 shows more details about this process. We used a_i for a specific arc. Each arc got assigned a sign: positive for convex and negative for concave as shown in Figure 24 b-).

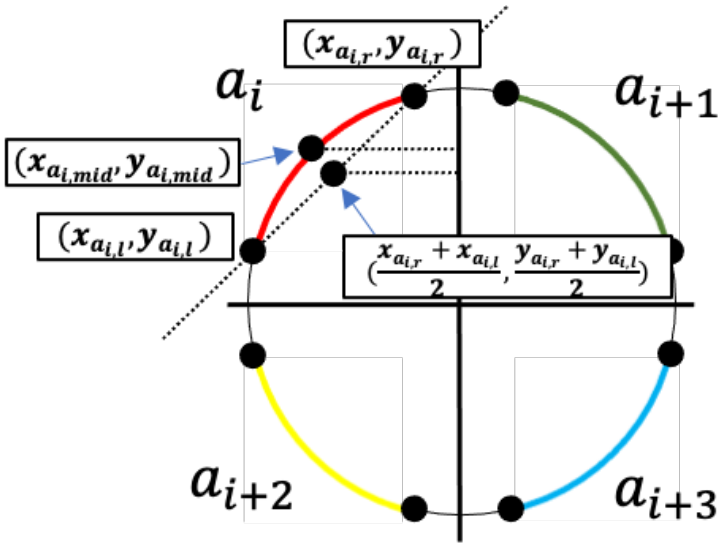

Figure 25. Convexity-concavity calculation

The convexity-concavity calculation is shown in Equation 10.

$$conv(a_i) = \begin{cases} True\ , & \frac{y_{a_{i,r}} + y_{a_{i,l}}}{2} > y_{a_{i,mid}} \\ False\ , & otherwise \end{cases} \tag{10}$$

where $y_{a_{i,mid}}$ is the midpoint of the arc, $y_{a_{i,r}}$ and $y_{a_{i,l}}$ are the y positions of the right and left endpoints of the line

The region assignment was done with the gradient and convexity signs. The overall assignment rules are shown in Equation 11.

$$r(a_i) = \begin{cases} 1, & if\ \left(a_i \in A_{neg}\right)\ and\ \left(conv(a_i) = True\right) \\ 4, & if\ \left(a_i \in A_{pos}\right)\ and\ \left(conv(a_i) = True\right) \\ 2, & if\ \left(a_i \in A_{pos}\right)\ and\ \left(conv(a_i) = False\right) \\ 3, & if\ \left(a_i \in A_{neg}\right)\ and\ \left(conv(a_i) = False\right) \end{cases} \tag{11}$$

After final region assignment, we started to merge these arcs. To associate each arc with another correct arc, we used some end point proximity thresholds. We summarized the rules in Equation 12. You can see each arc region from Figure 24 c-). We merged the neighbor arcs, i.e. 1-2, 1-4, 3-4 and 2-3.

$$m(a_i, a_j) = \begin{cases} True, & if\ (r(a_i, a_j) = (1,2)\ and\ |y_{a_{i,l}} - y_{a_{j,l}}| < \emptyset_{yv}\ and\ |x_{a_{i,l}} - x_{a_{j,l}}| < \emptyset_{xv}) \\ True, & if\ (r(a_i, a_j) = (1,4)\ and\ |y_{a_{i,r}} - y_{a_{j,l}}| < \emptyset_{yh}\ and\ |x_{a_{i,r}} - x_{a_{j,l}}| < \emptyset_{xh}) \\ True, & if\ (r(a_i, a_j) = (4,3)\ and\ |y_{a_{i,r}} - y_{a_{j,r}}| < \emptyset_{yv}\ and\ |x_{a_{i,r}} - x_{a_{j,r}}| < \emptyset_{xv}) \\ True, & if\ \ (r(a_i, a_j) = (2,3)\ and\ |y_{a_{i,r}} - y_{a_{j,l}}| < \emptyset_{yh}\ and\ |x_{a_{i,r}} - x_{a_{j,l}}| < \emptyset_{xh}) \end{cases} \quad (12)$$

$\emptyset_{xv}$, $\emptyset_{yv}$, $\emptyset_{xh}$ and $\emptyset_{yh}$ are the thresholds for distances between the endpoints of the candidate arcs to merge. In these thresholds, v refers to the vertical threshold (for arc pairs in regions 1-4 and 2-3) and h does the horizontal threshold (for arc pairs in regions 1-2 and 4-3). Additionally, x and y show the axis direction for the thresholds. In this problem, because of the water presence, arcs from regions 2 and 3 may not be visible (under the water) or may be very small. To solve this issue, we proposed a hierarchy-based order in merging these arcs. We started with arcs from region 1 and 4. Then, we added the arcs from regions 2 and 3. After each merge, arc sets turned into partial or full ellipses. We summarized this process with a visualization in Figure 26. From left to right, we showed the raw image (a), edges (b), positive gradient points (c), negative gradient points (d) and arcs with different colors for each region (e).

Figure 26. Arc extraction and merge processes

- **Constrained Least Squares Solution for the Ellipse:**

After finding the partial or full ellipse points in the image, next, we fit an ellipse with these data points. We used the numerically stable least-squares solution for the ellipse fit by Halir and Flusser (1998). This is a simple and fast technique. Overall, there are five steps in this process.

- **Step 1: Define the conic equation:**

  As shown earlier in Equation 7, the ellipse equation can be written as a conic equation with an additional constrained. In this equation, a, b, c, d, e and f are coefficients.

$$F(x,y) = ax^2 + bxy + cy^2 + dx + ey + f = 0 \quad (13)$$

with ellipse constraint

$$b^2 - 4ac > 0 \quad (14)$$

  Here, (x, y) are the data points that satisfy the equation and therefore lie on the ellipse. F(x, y) measures the distance of a given data point (x_n, y_n) from the ellipse.

- **Step 2: Writing the equation as a vector multiplication:**

  To solve the ellipse equation, we use all the data points from the merged arcs. For this reason, we rewrote the Equation 13 in vector multiplication form and got Equation 15.

$$F(x) = x.a = 0 \quad (15)$$

where

$$a = [a, b, c\ , d, e, f]^T$$
$$x = [x^2, xy, y^2, x, y, 1] \quad (16)$$

Both a and x are vectors. Similarly, like before, [a, b, c, d, e, f] are the parameters to learn from the data points.

- **Step 3: Minimize the cost function:**

  We minimized the sum of squared distances between the data points. The cost function is given in Equation 17.

$$\min \sum_{i=1}^{N} \left(F(x_i)\right)^2 = \min_a \sum_{i=1}^{N} (a.x_i)^2 \tag{17}$$

- **Step 4: Apply the ellipse constraint:**

  As shown earlier, there is an added term (Equation 14) to the ellipse equation. To make it simple, we assumed a positive value:1 for the constraint.

$$b^2 - 4ac = 1 \tag{18}$$

  With this simplifying assumption, the minimization problem became like this:

$$\begin{aligned} &\min_a ||Da||^2 \\ &\text{subject to } a^T Ca = 1 \end{aligned} \tag{19}$$

  where D is a nx6 matrix for n data points and its columns correspond to a, b, c, d, e and f coefficients of the ellipse equation from Equation 13. C is a 6x6 matrix and it satisfies the condition in Equation 19. D and C are shown below.

$$D = \begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ .. & .. & .. & .. & .. & .. \\ x_i^2 & x_iy_i & y_i^2 & x_i & y_i & 1 \\ .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. \\ x_n^2 & x_ny_n & y_n^2 & x_n & y_n & 1 \end{bmatrix} and\ C = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{20}$$

- **Step 5: Solve the equation with Lagrange multipliers:**

We used the Lagrange multipliers to solve the Equation 19. With this change of notation, the equation was written like below.

$$Sa = \lambda Ca$$

$$\text{subject to } a^T Ca = 1 \tag{21}$$

$$\text{where } S = D^T D$$

At the end, the constrained problem was solved by finding the minimum positive eigenvector a_k and its corresponding eigenvalue λ_k from Equation 21.

- **Calculating the water depth:**

In the previous section, we calculated the equation for an ellipse for each tire wheel. As the last step, the water depth was as the ratio of the wheel under the water. If there are more than one tire in the image, an average value is calculated for each vehicle. Figure 27 shows an example for this calculation. The ration d/h yields the water depth ratio for the given example tire image. When there is a rotation angle in the ellipse, the ratio d/h still works.

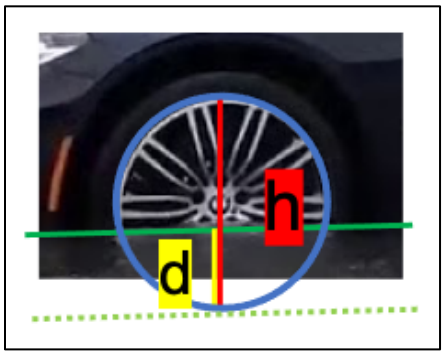

Figure 27. Tire water depth ratio

4.5 Data

In this project, we used both real and synthetic images and trained object detection and semantic segmentation networks. We used the synthetic data due to the lack of available labeled images in this problem. With synthetic data generation, we were able to create labeled images in large amounts. Additionally, we used real-world images collected from web and labeled them. For semantic segmentation, pixels were labeled and for object detection, bounding box labels were created. In the dataset, each image contains a single vehicle in a flood scene. For object detection, labels were the boxes of the tires of vehicles. For semantic segmentation, tire wheels and water labels were labeled in pixel level.

We used 663 images in the semantic segmentation task. The dataset was split into training, validation, and test subsets having 574, 44, and 45 images respectively. We used a mix of 425 synthetic and 149 real images in the training set. We used only real images (no synthetic images) in the validation and test sets to make sure the models were validated and test with real-world scenarios. The real images in all sets were collected from flood images on the web and synthetic images were created using the Unity software (Unity 2021). More details about the synthetic images are given later in this section.

For the object detection task, similarly, we had the training and validation split. In this task, we used additional images in these sets to learn this task better. Usually, creating labels for object detection is quicker and less costly than labeling pixels for semantic segmentation. Here, we used 303 vehicle pictures from the Compcar dataset by Yang et al. (2015) and 646 new flood images from the web. The images from the CompCar dataset don't contain any floodwater. To make these images similar to having flood conditions (tires submerged in water), we applied

some data augmentation on them by randomly cropping the bottom parts of the images. In total, we used 1,567 with a 70-30% split for training and validation. The test dataset is the same as in the semantic segmentation task.

4.5.1 Synthetic Data

In this project, we used synthetic images of flood scenes. These images were created using the Unity software (Unity 2021). To create a flood scene, we first created a small city and then added water to the streets to create the flood effect. Some sample images are shown in Figure 28. Figure 28 a) shows the created city without water added. Figure 28 b) shows the city with water. The images we used contained water in them. To understand the water depth in the images, we used vehicles as helper objects. In the synthetic dataset, we used 17 different 3D car models from Unity. Overall, this gave a total of 425 synthetic images (25 images for each 3D model).

One of the advantages of using synthetic data generation is its efficiency and speed in creating training samples. We automated the data generation process. We first randomly placed the vehicle at a location, then added random offsets to the camera. Camera height was fixed at 1.7 units and the camera was facing straight without tilt angle. Offsets in the X direction were sampled from a uniform distribution between 5-7 units and similarly Z direction offsets were sampled from uniform distribution between -3 and 3. The axis system was defined like this: Y axis pointing upwards, z pointing the left direction and x going towards this page.

To add more randomization to this process, we applied random horizontal flips and rotations. The rotation angles were sampled from uniform distribution between -30 and 30 degrees. Rotation angles were kept within the +30 and -30 degrees to make sure we had side view images of vehicles.

As another randomized operation, we also changed the water depth randomly. The values for the water depth in the program was selected uniformly from these values: 0.05, 0.1, 0.15, 0.2, 0.25 and 0.3 units.

Overall, these randomized operations helped us create randomly generated data. Figure 28 c) and d) show a sample picture with a vehicle in it (without and with water). Figure 28 e) shows pixel labels. These labels were automatically extracted from Unity. Bounding box labels for object detection were labeled manually from the generated images. At the end, we generated 425 images from 17 different vehicles (25 images per vehicle). Some additional images with the different vehicles and from different parts of the city are show in Figure 33 and Figure 34.

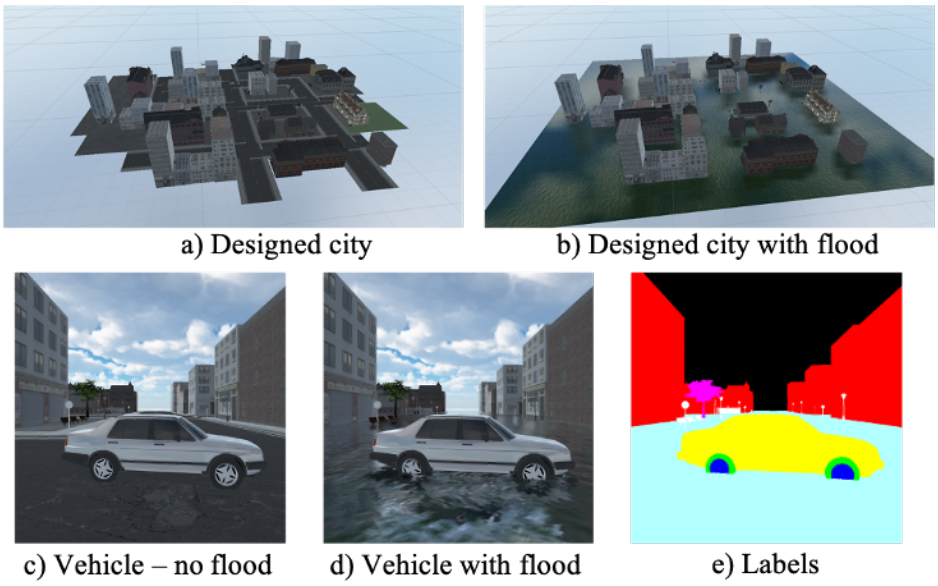

Figure 28. Synthetic image examples

The breakdown of the dataset in terms of training, validation, test images and real-synthetic distinction is shown in Table 9.

Table 9. Dataset breakdown

Model	**# of Training images**		**# of Validation images**		**# of Test images**	**Total # of images**	
	Synthetic	**Real-world**	**Synthetic**	**Real-world**	**Real-world**	**Synthetic**	**Real-world**
Semantic Segmentation	425	149	-	44	45	425	238
YOLO	298	588 flooded images + 213 CompCar images	127	251 flooded images + 90 CompCar images	45	425	884 flooded images + 303 CompCar images

4.6 Results

In this section we summarize results on validation and test datasets and give details about the training process. The dataset breakdown was previously shown in Table 9.

4.6.1 Training

We used multiple models in this problem. For the initial semantic segmentation task, we used the FCN architecture with three different base networks: VGG-16, MobileNet and ResNet 101. We learned to segment images into background, wheel and water classes in pixel level. The shape of the input images was 384 x 512. For object detection a YOLO detector was trained. This detector produced the bounding boxes for tires in the images. In addition to these, we used another semantic segmentation network to improve the initial segmentation results. This network used two inputs: The raw tire image from YOLO and the corresponding initial segmentation patch (at the same location). These two images were resized to 128x128. This network learned to fix some of the initial segmentation errors. Due to tires being small objects compared to the rest of the picture, this improvement in general gave more accurate boundaries for wheels and water in the images. During the training of this network, we increased the weights of the pixels that belong to a band along the border of wheel and water. The water depth calculation part didn't use any trained network. For that part, we selected parameters and thresholds of the algorithm using the validation images.

In order to make sure the models generalize well and do not overfit, we created multiple random splits of the dataset. This produced the training, validation and test sets. Additionally, we used the synthetic images only in the training set and validate/test with real images only to create a robust system.

4.6.2 Hyper-parameters and Validation Results

We used 45 validation images to decide hyperparameters and thresholds/parameters for the water depth prediction method. Table 10 shows the hyperparameters for neural networks. We report the optimizers, learning rates, batch size and overall training epochs.

Table 10. Neural Network Hyper-parameters

Model	Training optimizer	Learning rate	Batch size	Epochs
FCN-VGG16	Adam	0.0001	8	6
FCN-MobileNet	Adagrad	0.0001	4	5
FCN-ResNet101	Adagrad	0.0001	4	11
U-Net for FCN-VGG16	Adagrad	0.001	4	7
U-Net for FCN-MobileNet	Adagrad	0.001	8	6
U-Net for FCN-ResNet101	Adagrad	0.001	4	7

We used multiple parameters in the ellipse fit method. The processes for this method were shown earlier in Figure 23. We had bilateral and canny edge filters. Bilateral filter used σ_{color}=75, σ_{space}=150 with diameter of neighborhood 9. Fixed thresholds didn't work well for the canny edge detector. It caused too many or too few edge pixels in different cases. We solved this problem with adaptive thresholds using the median values of intensities. Equation for this is shown below in Equation 22. We used σ = 0.75.

$$lower\_threshold = \max(0, (1-\sigma) * median(grayscale\_image))$$
$$upper\_threshold = \min(255, (1+\sigma) * median(grayscale\_image)) \quad (22)$$

Additionally, we used some thresholds in merging of arc pieces to build bigger structures like 2-piece (half), 3-piece or 4-piece (full) ellipses (Equation 12). These thresholds values were used for them: $\emptyset_{xv}$=8, $\emptyset_{yv}$=25, $\emptyset_{xh}$=30, $\emptyset_{yh}$=8. These are in pixel units.

We summarize the validation results below. We used precision, recall and F1 scores. We start with the semantic segmentation results. These scores were calculated per pixel using the background, wheel and water classes on 384x512 images. They are shown in Table 11.

Table 11. Semantic Segmentation Results on validation images (384x512)

Model	Precision	Recall	F1	Accuracy
FCN-VGG16	0.85	**0.9**	0.88	0.93
FCN-MobileNet	0.89	0.84	0.86	0.93
FCN-ResNet101	**0.94**	0.89	**0.91**	**0.95**

FCN with ResNet101 base network achieved the highest scores except the recall. On the recall, FCN with VGG-16 did slightly better.

Additionally, we present the water depth calculation errors for the validation images in Table 12. We ran two types of experiments. In the first type, we used the algorithm outlined in Figure 23. In the second type, we used only the refined segmentation result to fit the ellipses. This corresponds to the top part of Figure 23. We predicted the water depth ratio using the d/h ratio where d is the water depth and h is the height if the tire wheel. Overall, this produces values between 0 and 1.

Table 12. Results of ellipse fit methods on validation set

Model	Ellipse fit method	Mean Absolute Error (MAE)
FCN (VGG16) + U-Net	Arc-based + semantic segmentation fit	0.119
	Only semantic segmentation fit	0.139
FCN (MobileNet) + U-Net	Arc-based + semantic segmentation fit	0.203
	Only semantic segmentation fit	0.223
FCN (ResNet101) + U-Net	**Arc-based + semantic segmentation fit**	**0.115**
	Only semantic segmentation fit	0.129

FCN with ResNet101 achieved the best scores in both of the ellipse fit methods with 0.115 and 0.129 Mean Absolute Errors (MAEs). Similar to the semantic segmentation results, VGG16 based model came second and MobileNet based model came last.

4.6.3 Test Results

Like the validation set, we used another 45 real-world images in the test set. We ran the same tests with some additional analysis. Table 13 shows the semantic segmentation results on 384x512 sized test images. Same as in the case of validation set, FCN with ResNet101 achieved the highest scores in F1 and accuracy. FCN with VGG16 got slightly better score in Recall.

Table 13. Semantic Segmentation Results on test images (384x512)

Model	Precision	Recall	F1	Accuracy
FCN-VGG16	0.82	**0.9**	0.86	0.93
FCN-MobileNet	0.86	0.85	0.85	0.93
FCN-ResNet101	**0.92**	0.88	**0.90**	**0.95**

Additionally, we presented the results of semantic segmentation refinement. This task took two inputs: The raw tire images from YOLO detector and the corresponding initial segmentation results. At the end, it improves the initial segmentation results. Table 14 summarizes the results on test dataset for 128x128 images. We used three different U-Net like architectures for each semantic segmentation network (FCN-VGG16, FCN-MobileNet and FCN-ResNet101). The refinement networks improved the initial segmentation results in all cases.

Table 14. Impact of the refinement network on test tire images (128x128)

		Precision		Recall		F1		Accuracy	
Model	**Class**	**Before**	**After**	**Before**	**After**	**Before**	**After**	**Before**	**After**
FCN (VGG16) + U-Net	Background	0.96	0.97	0.95	**0.99**	0.96	0.98	0.94	0.96
	Water	0.92	0.93	0.89	0.96	0.90	0.94		
	Wheel	0.89	**0.97**	0.93	0.92	0.91	0.94		
FCN (MobileNet) + U-Net	Background	0.93	0.94	0.92	0.95	0.92	0.94	0.89	0.92
	Water	0.83	0.91	0.89	0.90	0.86	0.90		
	Wheel	0.86	0.87	0.83	0.86	0.84	0.87		
FCN (ResNet101) + U-Net	Background	0.96	**0.99**	0.98	**0.99**	0.97	**0.99**	0.96	**0.98**
	Water	0.92	**0.98**	0.96	**0.97**	0.94	**0.98**		
	Wheel	0.97	**0.97**	0.91	**0.98**	0.94	**0.97**		

Water depth ratio prediction errors on the test set are presented in Table 15. Same as in the validation dataset, FCN with ResNet101 achieved the best scores in both of the ellipse fit methods with 0.038 and 0.040 Mean Absolute Errors (MAEs). For the best score, differently from the validation results, using only the semantic segmentation result to fit ellipses resulted in a slightly better result. Although MobileNet and VGG16 based models did better with the hybrid (Arc-based + semantic segmentation) method.

Table 15. Results of ellipse fit method

Model	**Ellipse fit method**	**Mean Absolute Error (MAE)**
FCN (VGG16) + U-Net	Arc-based + semantic segmentation fit	0.044
	Only semantic segmentation fit	0.061
FCN (MobileNet) + U-Net	Arc-based + semantic segmentation fit	0.089
	Only semantic segmentation fit	0.168
FCN (ResNet101) + U-Net	Arc-based + semantic segmentation fit	0.040
	Only semantic segmentation fit	**0.038**

Lastly, we analyze the distribution of test errors. We picked the top 3 results and plotted them in Figure 29. The first column shows the error histogram, the second column is the histogram of

Mean Absolute Errors for different true water depth ratios and the last column is a scatter plots of errors with respect to true water depth ratios. The error histograms (first column) and error scatter plots (third column) show that VGG16 based method produced errors with larger spread. Errors were approximately between -0.2 and +0.2 Additionally, in the MAE histogram (second column), the same model produced the largest MAE (0.09) in its last bin. The last bin corresponds to the images with the highest level of water. For the same column, ResNet based methods stayed within the 0.03-0.05 range for all bins. The scatter plots (third column) looked similar for ResNet based methods, but the VGG-based method produced more errors beyond the +=0.01 range. Lastly, for all three methods, most of the negative errors (prediction bigger than true value) occurred when the true water depth was greater than 0.4.

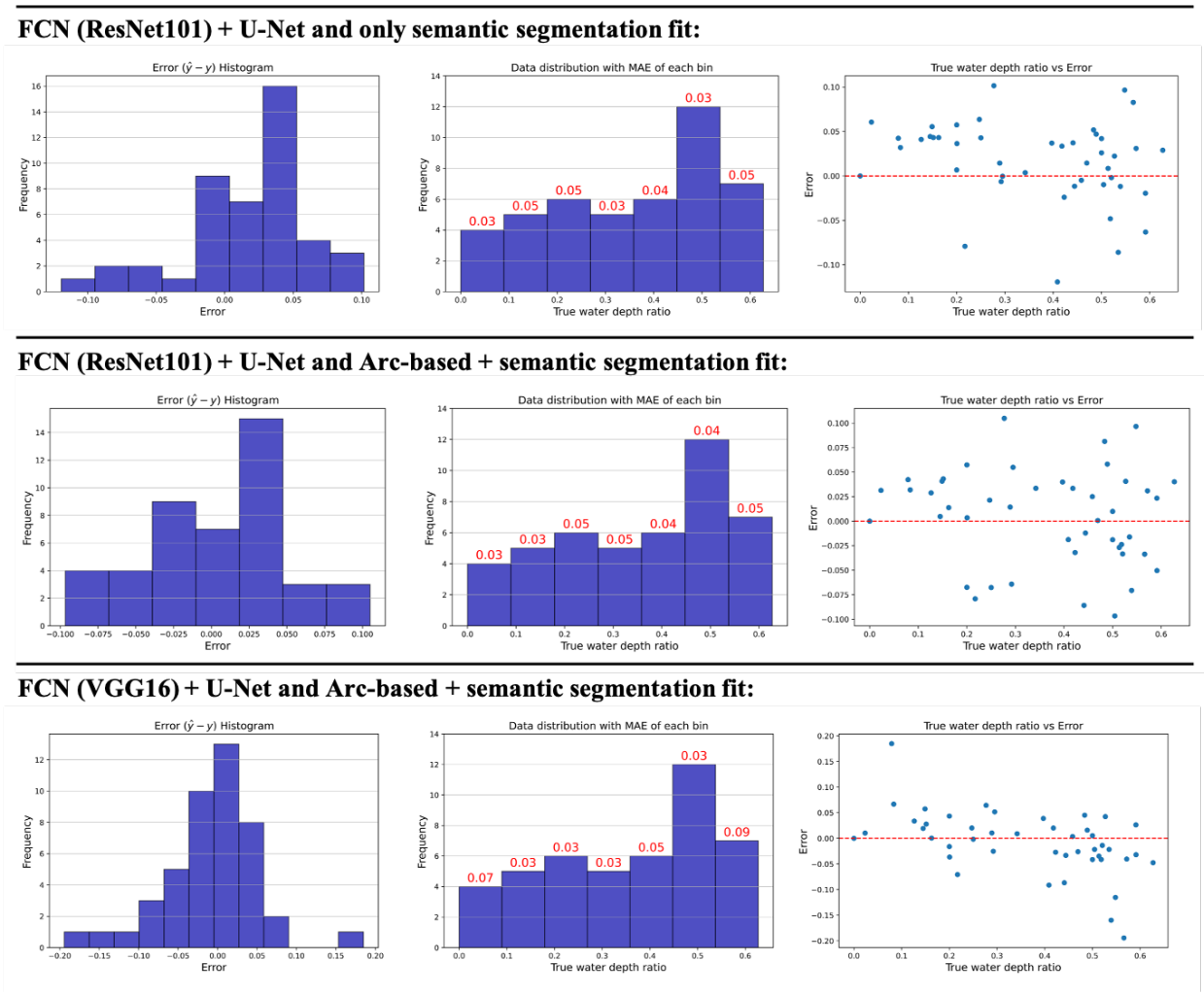

Figure 29. Error Analysis of top three results

4.6.4 Sensitivity Analysis

In this section, we provide a sensitivity analysis of the proposed method. This analysis investigates the performance of the model in terms of the Mean Absolute Error (MAE) as vehicle view angle and true water depth ratio are varied. In this work, we used a controlled environment in Unity and changed the view angle of vehicles and depth of the water.

The tested levels of parameters are as follows:

- **View angles in degrees:** [0, 10, 20, 30, 40, 50, 60]

- **True water depth ratios:** [0.17, 0.26, 0.38, 0.50, 0.61]

These levels correspond to 35 possible scenarios for each image. View angles are limited to 60 degrees at most because object detector starts to fail in detecting tires beyond 60 degrees. In total, 140 images were created for this analysis – four samples (four different vehicles) per scenario. Figure 30 shows a sample vehicle with different view angles. The water depth ratio in that figure is 0.17. In the view angle selection, angles larger than 60 degrees are not used as they don't display the tires for the model to work with. For water depth ratio, multiple values ranging from 0.17 to 0.61 were used to see the impact on the performance of the model. The images used in this analysis were not used in the training of the model. Figure 31 shows a heatmap for MAE obtained from a trained model explained previously (i.e., ResNet and Arc-based + semantic segmentation fit). For different view angles and true water depth ratios, the MAE for different scenarios is shown in the figure where darker color corresponds to poorer performance.

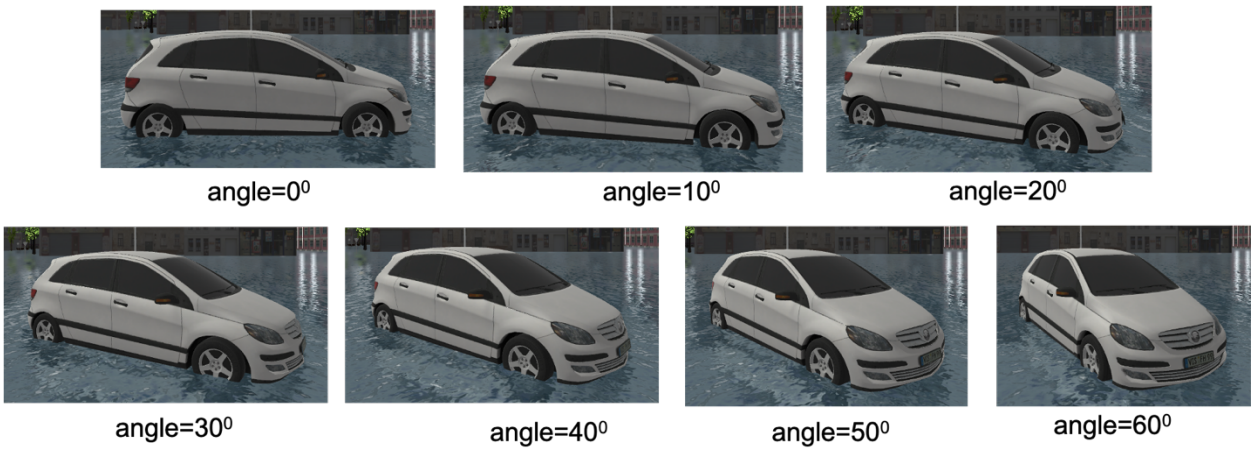

Figure 30. Sensitivity Analysis

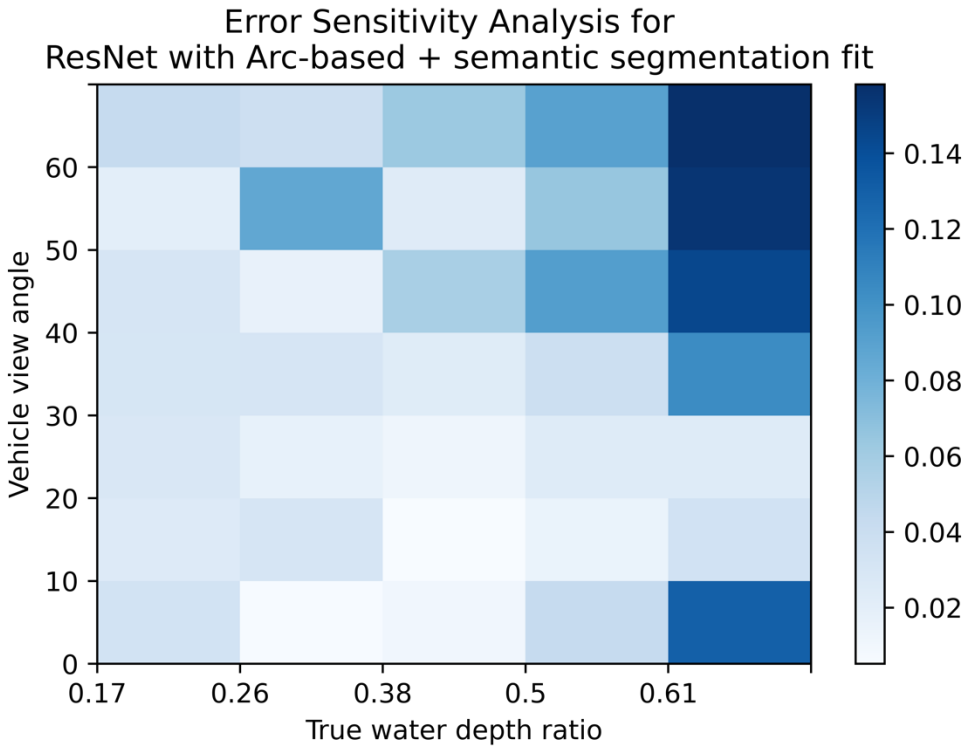

Figure 31. Sensitivity Analysis

We can easily see that absolute errors usually increase as the water depth ratio increases. Especially when the water depth ratio is 0.61 and view angles are larger than 30 degrees, errors are large. These are all expected as the model heavily relies on the side view angles. We observe that the model produces significant errors when the true water depth ratio is 0.61, especially at skewed view angles.

4.7 Discussion

One of the advantages of the proposed method is that the errors can be attributed to different parts of the model overall. For example, sematic segmentation model can have misclassifications in cases when there is strong reflection on the water. Although the semantic segmentation improvement model can learn to fix them, sometimes some errors get passed to the next steps and cause errors for the water depth predictions at the end. Additionally, the ellipse fit

method can struggle when there are not enough data points to fit on. These can happen when the water level is high causing tires to be submerged in the water, when the tire is occluded by another object or when the image is low-resolution. These problems can cause the algorithm to find the arcs corresponding to lower parts of ellipses. Some good and bad fits are shown in Figure 32. This explainable nature of our method gives us advantages over black-box systems such as the Mask-RCNN models proposed by Meng et al. (2019) and Chaudhary et al. (2019). Lastly, the proposed method can make predictions without knowing the actual size of the tires and the vehicle type. The method by Park et al. (2021) depends on correct alignment between 2D images and 3D models that are sensitive to different vehicle types.

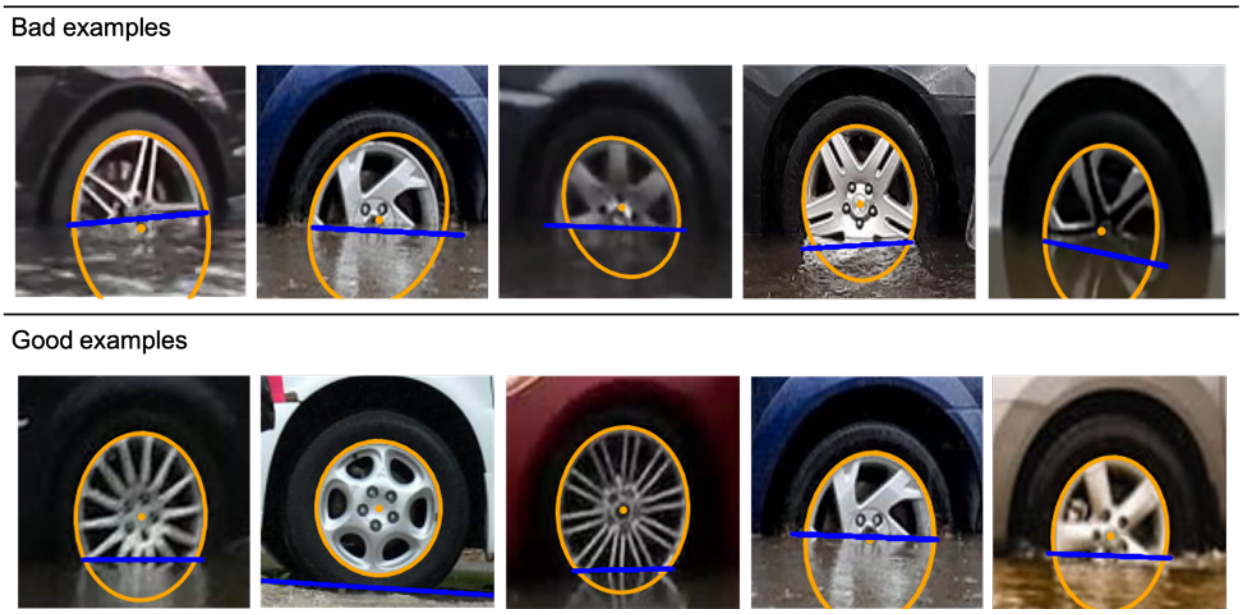

Figure 32. Some example results

4.8 Conclusion and Future Work

In this chapter of the dissertation, we proposed a method to calculate water depth in flood images using a mix of image processing and deep learning methods. We calculated the water depth as a ratio of the vehicle tire wheel under the water. Our method used multiple models to

learn different tasks. Although this may be seen as a difficulty in terms training compared to end-to-end trainable systems like Fast-RNN models, this allowed us to explain our results to a certain degree. For example, we can go back and find the source(s) of errors within the workflow of our methods and target some failing processes in the future.

Due to the lack of labeled datasets, we labeled flood images collected from web and created synthetic images using the Unity Software. This work is the first flood depth prediction method that learned from synthetically generated data. We created a total of 425 synthetic images of 17 distinct vehicles.

Our workflow included different components: semantic segmentation, segmentation refinement, object detection and water depth calculation. This multi-model system achieved as low as 0.038 Mean Absolute Error (MAE) in the test set of 45 real-world images. Our analysis showed that in the test set, ResNet-based method didn't need the arc discovery and merge technique, although it helped achieve better scores with the other base networks.

Our method made the following assumptions. We assumed that the vehicles are expected to travel at low speeds or stand-still. We made this assumption to correctly predict water edge on the tires. Otherwise, water splashes make this process difficult. This assumption may limit the overall applicability of this method, but this method can be applied at certain locations where the vehicles stop or move slowly such as signalized intersections, or local roads with parked cars.

As future work, learning the 3D representations of the vehicles and the scenes will be investigated. This can help produce more accurate predictions and extend the predictions to understanding the extend of floodwater such as closed lanes and intersections.

CHAPTER 5

CONCLUSIONS

This dissertation presented multiple studies to solve prediction problems related to floodwater on roadways. Mainly, two studies were presented. In the first study, the image classification and semantic segmentation problems were solved. A comparison between "classical" feature extraction and deep feature extraction methods were presented. In the second study, a novel floodwater depth prediction method was proposed. This method not only predicted the floodwater depth, but also semantically segmented the given images. This section summarizes these studies and their results.

Chapter 3 covered a set of methods to classify and semantically segment images with floodwater in them. In this work, image classification and semantic segmentation tasks were handled separately. For image classification, multiple feature extraction and predictors were developed and compared. For feature extraction, two classical approaches: Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP) and the deep feature extraction by a pre-trained VGG-16 network were used. As classifiers, decision trees, k nearest neighbors and logistic regression methods were used with these extracted features. Overall, 9 different combinations were developed and tested. For semantic segmentation, two main approaches were used: Superpixel based segmentation and Fully Convolutional Networks (FCNs). In the superpixel case, the same types of classifiers were trained to classify superpixels this time. These methods were trained and tested with hand-labeled images collected via web search. At the end, the pre-trained feature extraction with logistic regression achieved the best scores in

classification problem. For the semantic segmentation problem, the FCN model achieved the best performance.

Chapter 4 proposed a novel approach to predict floodwater depth on roadways. This method was implemented as a combination of multiple components. Each component was responsible for a certain task. The following were the four components: Initial semantic segmentation of the flood image, Object detection to find tires in the image, Semantic segmentation refinement network to improve initial segmentation results and floodwater depth prediction via fitting ellipses on wheel parts of tires, line fitting on the water edges and calculating the ratio of wheels under the water. The proposed method had certain advantages compared to the other methods in the literature. First, it was explainable to a certain degree. For example, we could study the outcome of each component. Second, the proposed water depth metric allowed us to make calculations using ratios. With this, prior information about the vehicle or 3D model match steps were not needed. Third, we utilized synthetic data generation for this problem. When needed, we could create even bigger datasets by adding more vehicle types and changing the environment. Overall, this method was able to predict the water depth with a Mean Absolute Error of 0.038 on the test images. Although not tested yet, the proposed methods to fit ellipses on tire wheel parts can be extended to regular vehicle images without flood to locate vehicle tires precisely for different tasks.

REFERENCES

Chaudhary P, D'Aronco S, Moy de Vitry M, Leitao JP, and Wegner, JD (2019) Flood-Water Level Estimation From Social Media Images. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci. IV-2/W5: 5-12

Chen LC, Papandreou, G, Schroff, F, Adam, H (2017) Rethinking atrous convolution for semantic image segmentation. In arXiv:1706.05587

Cohen S, Brakenridge GR, Kettner A (2018) Estimating floodwater depths from flood inundation maps and topography. Journal of the American Water Resources Association 54(4): 847-858.

Cohen S, United States Flood Inundation Mapping Repository. Available: https://sdml.ua.edu/projects/usfimr Accessed April 21, 2021

Dalal N and Triggs B (2005) Histograms of oriented gradients for human detection. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005

Dong H, Sun G, Pang WC, Asadi E, Prasad DK, Chen IM (2018) Fast Ellipse Detection via Gradient Information for Robotic Manipulation of Cylindrical Objects. IEEE Robotics and Automation Letters 3(4): 2754-2761.

Eldesokey A, Felsberg M, Khan F (2017) Ellipse detection for visual cyclists analysis in the wild. In: Computer Analysis of Images and Patterns (CAIP), 2017, 319-331.

European Comission Copernicus Emergency Management Service (2020). Available: https://emergency.copernicus.eu/ Accessed April 21, 2021

Feng Y, Brenner C, Sester M (2020) Flood severity mapping from Volunteered Geographic

Information by interpreting water level from images containing people: A case study of Hurricane Harvey. ISPRS J. Photogramm. Remote Sens. 2020, 169, 301-319.

Fornaciari M, Prati A, Cucchiara R (2014) A fast and effective ellipse detector for embedded vision applications. Pattern Recognition, 47(11):3693-3708.

Gao N, Shan Y, Wang Y, Zhao Z, Yu Y, Yang M, and Huang K (2019) Ssap: Single-shot instance segmentation with affinity pyramid. In ICCV, 2019

Geetha M, Manoj M, Sarika AS, Mohan M, Rao SN (2017) Detection and estimation of the extent of flood from crowd sourced images. In: 2017 International Conference on Communication and Signal Processing (ICCSP), 2017, 603-608.

Girshick R (2015) Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision, 2015, 1440-1448.

Halir R, Flusser J (1998) Numerically Stable Direct Least Squares Fitting of Ellipses. Proc. Sixth Int Conf. Computer Graphics and Visualization, 1998, 125-132.

Hawker L, Bates P, Neal J, Rougier J (2018) Perspectives on Digital Elevation Model (DEM) Simulation for Flood Modeling in the Absence of a High-Accuracy Open Access Global DEM. Frontiers in Earth Science 6:233.

He K, Zhang X, Ren S, Sun J (2016) Deep Residual Learning for Image Recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, 770-778.

He K, Gkioxari G, Dollár P, and Girshick R (2017) Mask R-CNN. In IEEE International Conference on Computer Vision (ICCV), 2017

Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://arxiv.org/abs/1704.04861

Iwahashi M, Udomsiri S (2007) Water Level Detection from Video with Fir Filtering. In: 16th International Conference on Computer Communications and Networks, 2007, 826-831.

Jamil AA, Hussain F, Yousaf MH, Butt AM, Velastin SA (2020) Vehicle Make and Model Recognition Using Bag of Expressions. Sensors (Basel), 20(4): 1033

Kharazi BA and Behzadan AH (2021) Flood depth mapping in street photos with image processing and deep neural networks. Computers, Environment and Urban Systems, 88

Kettner A and Brakenridge R (2021) The Dartmouth Flood Observatory (DFO). Available: http://floodobservatory.colorado.edu/ Accessed April 21, 2021

Konadu DD, Fosu C (2007) Digital Elevation Models and GIS for Watershed Modelling and Flood Prediction – A Case Study of Accra Ghana, 2007, In: Appropriate Technologies for Environmental Protection in the Developing World, 2007, 325-332

Krizhevsky A, Sutskever I, and Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 2012, 1097-1105

Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, and Berg AC (2016) SSD: Single shot multibox detector. In ECCV, 2016

Lee HJ, Ullah I, Wan W, Gao Y, Fang Z (2019) Real-time vehicle make and model recognition with the residual SqueezeNet architecture. Sensors, 19: 982

Lin F, Chang W, Lee L, Hsiao H, Tsai W, Lai J (2013) Applications of Image Recognition for Real-Time Water Level and Surface Velocity. In: IEEE International Symposium on Multimedia, 2013, 259-262

Lopez-Fuentes L, van de Weijer J, Bolaños M, Skinnemoen H. (2017) Multi-modal deep learning approach for flood detection. In: Working Notes Proceedings of the MediaEval 2017 Workshop CEUR-WS, 2017, 1-3

Manzoor MA, Morgan Y, Bais A (2019) Real-Time Vehicle Make and Model Recognition System. Machine Learning and Knowledge Extraction, 1: 611–629

Meng Z, Peng B, Huang Q (2019) Flood depth estimation from web images. In: Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Advances on Resilient and Intelligent Cities, 2019, 37-40

Nair BB, Rao S (2016) Flood water depth estimation — A survey. In: IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2016, 1-4

Nair V and Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In ICML, 2010.

Neussner O, Obermaier I, Sanchez A (2012) Application of a digital elevation model for flood modeling for the Pagsangaan River in Leyte. In: 1st Philippine Geomatics Symposium (PhilGEOS), 2012, 1-7

Park S, Baek F, Sohn J, Kim H (2021) Computer Vision–Based Estimation of Flood Depth in Flooded-Vehicle Images. Journal of Computing in Civil Engineering 35(2)

Redmon J, Farhadi A (2018) Yolov3: An incremental improvement. https://arxiv.org/abs/1804.02767

Ronneberger O, Fischer P, Brox T (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, 234-241.

Sarp S, Kuzlu M, Cetin M, Sazara C, Guler O (2020) Detecting Floodwater on Roadways from Image Data Using Mask-R-CNN. In: International Conference on INnovations in Intelligent SysTems and Applications (INISTA), 2020, 1-6.

Sazara C, Cetin M, Iftekharuddin KM (2019) Detecting floodwater on roadways from image data with handcrafted features and deep transfer-learning. In: IEEE Intelligent Transportation Systems Conference (ITSC), 2019 804-809.

Shelhamer E, Long J, Darrell T (2015) Fully convolutional networks for semantic segmentation. In: Conference on Computer Vision and Pattern Recognition (CVPR), 2015, 3431-3440.

Shin I, Kim J, Lee SG (2008) Development of an internet-based water-level monitoring and measuring system using CCD camera. In: ICMIT 2007: Mechatronics, MEMS, and Smart Materials, 2008, vol: 67944Q

Siddiqui Z, Park U, Lee SW et al (2018), Robust powerline equipment inspection system based on a convolutional neural network. Sensors 18(11), 3837

Simonyan K, Zisserman A (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556

Sun B, Zhang C, Liu Z, Tian H, Zhang H (2014) Research on HVPL for visual detection of bicolor water level gauge. In: The 26th Chinese Control and Decision Conference (CCDC), 2014, 2094-2099.

Sun T, Zhang C, Li L, Tian B, Qian B, Wang J (2013) Research on image segmentation and extraction algorithm for bicolor water level gauge. In: 25th Chinese Control and Decision Conference (CCDC), 2013, 2779-2783.

Sweet WV, Park J. (2014) From the extreme to the mean: Acceleration and tipping points of coastal inundation from sea level rise. Earths Future 2(12):579-600.

Tan M, Pang R, Le QV (2019), EfficientDet: Scalable and efficient object detection. In: Conference on Computer Vision and Pattern Recognition (CVPR), 2020, 10781-10790.

Terti G, Ruin S, Anquetin S, Gourley JJ (2017) A Situation-Based Analysis of Flash Flood Fatalities in the United States. Bulletin of the American Meteorological Society 98(2):333-345.

Ultralytics (2020), Yolo v5 available: https://github.com/ultralytics/yolov5

Unity Software (2021) Available: https://unity.com/ Accessed April 21, 2021

Wang N and Yeung DY (2013) Learning a deep compact image representation for visual tracking. In Advances in neural information processing systems, 2013, 809-817.

Witherow MA, Elbakary MI, Iftekharuddin KM, Cetin M (2018) Analysis of Crowdsourced Images for Flooding Detection. In: European Congress on Computational Methods in Applied Sciences and Engineering, 2018, 140-149.

Witherow MA, Sazara C, Winter-Arboleda IM, Elbakary MI, Cetin M. (2019) Floodwater detection on roadways from crowdsourced images. Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization 7(5-6), 529-540.

Yang L, Luo P, Loy CC, Tang X (2015) A Large-Scale Car Dataset for Fine-Grained Categorization and Verification. In: Conference on Computer Vision and Pattern Recognition (CVPR), 2015, 3973–3981.

Young DS, Hart JK, Martinez K (2015) Image analysis techniques to estimate river discharge using time-lapse cameras in remote locations. Computers & Geosciences 76:1-10.

APPENDIX A

ADDITIONAL IMAGES FOR CHAPTER 4

Figure 33. Sample images of 17 different vehicle models on Unity

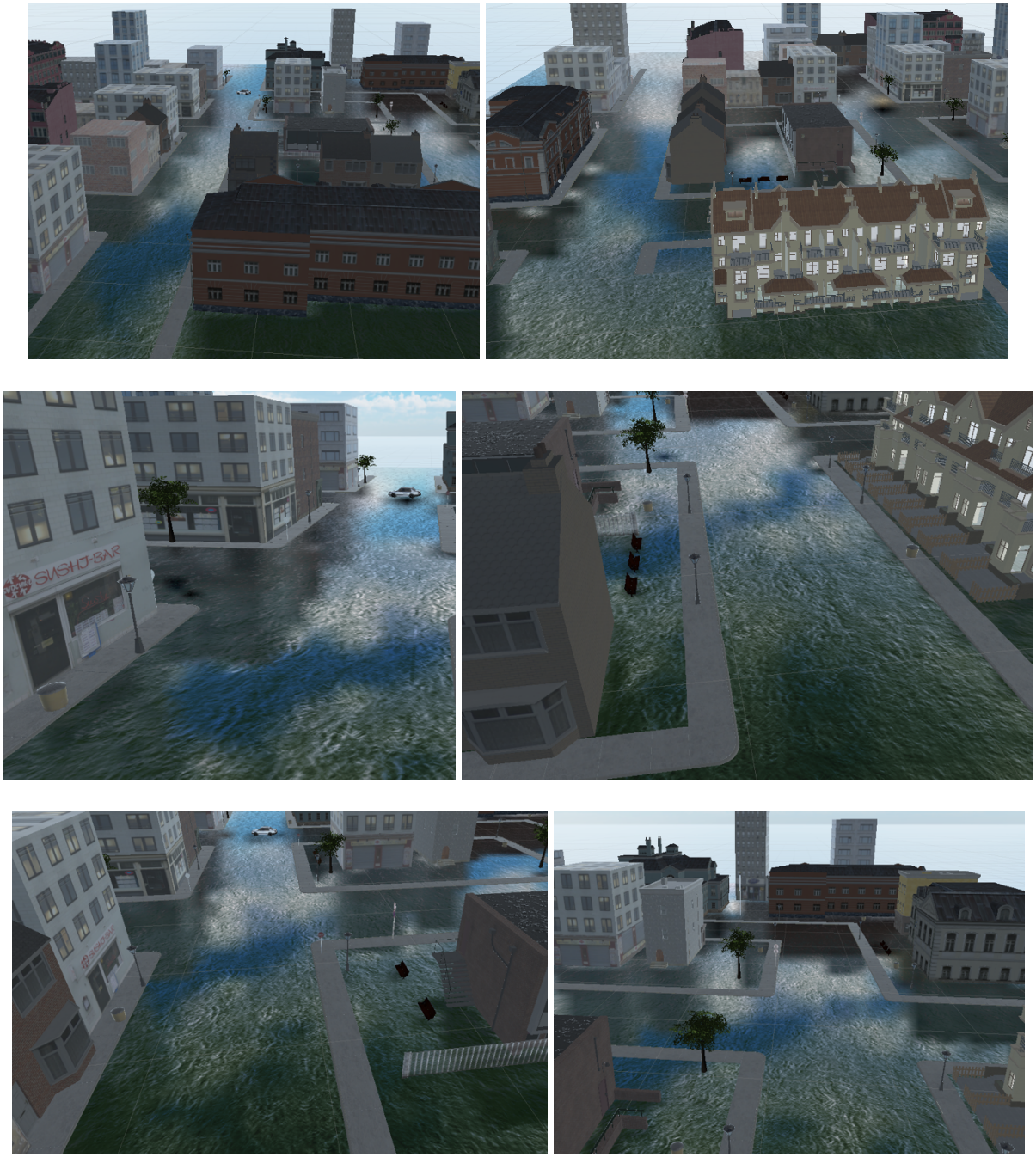

Figure 34. Some images from the synthetic environment

VITA

Cem Sazara

Biography

Cem Sazara is a Ph.D. candidate in the Computational Modeling and Simulation Engineering (CMSE) Department at Old Dominion University and an Applied Scientist at Amazon. He earned his bachelor's degree at Electrical-Electronics Engineering from Bogazici University. After that he worked in the industry as software engineer for two years. Then, he worked as a graduate assistant at Colorado State University – Pueblo and continued as research assistant at Old Dominion University throughout his doctoral studies.

Conference and Journal Papers

1- **Sazara C**, Bedoya-Valencia L (2015) Developing Dispatching Rules for a Dynamic Flexible Flow Shop Scheduling Problem at a Powder Coating Facility. In Industrial and Systems Engineering Research Conference, 1242-1248

2- **Sazara C**, Nezafat RV, Cetin M (2017) Offline reconstruction of missing vehicle trajectory data from 3D LIDAR. In 2017 IEEE intelligent vehicles symposium (IV), 792-797

3- Witherow MA, **Sazara C**, Winter-Arboleda IM, Elbakary MI, Cetin M, Iftekharuddin KM (2019) Floodwater detection on roadways from crowdsourced images. Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization 7(5-6), 529-540

4- **Sazara C**, Cetin M, Iftekharuddin KM (2019) Detecting floodwater on roadways from image data with handcrafted features and deep transfer-learning. In: IEEE Intelligent Transportation Systems Conference (ITSC), 2019, 804-809

5- Sarp S, Kuzlu M, Cetin M, **Sazara C**, Guler O (2020) Detecting Floodwater on Roadways from Image Data Using Mask-R-CNN. In: International Conference on INnovations in Intelligent SysTems and Applications (INISTA), 2020, 1-6

6- (Under review) **Sazara C**, Salahshour B, Cetin M, Iftekharuddin KM (2021) A Deep Learning Method for Floodwater Depth Prediction on Roadways from SideView Real and Synthetic Images of Vehicles. 2021