Application of Image Processing and Convolutional Neural Networks for Flood Image Classification and Semantic Segmentation

Jaku Rabinder Rakshit Pally
jpally@clemson.edu

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Part of the Computer and Systems Architecture Commons, Data Storage Systems Commons, and the Risk Analysis Commons

Recommended Citation
https://tigerprints.clemson.edu/all_theses/3662

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
Application of Image Processing and Convolutional Neural Networks for Flood Image Classification and Semantic Segmentation

Jaku Rabinder Rakshit Pally

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Part of the Computer and Systems Architecture Commons, Data Storage Systems Commons, and the Risk Analysis Commons
APPLICATION OF IMAGE PROCESSING AND CONVOLUTIONAL NEURAL NETWORKS FOR FLOOD IMAGE CLASSIFICATION AND SEMANTIC SEGMENTATION

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

by
Jaku Rabinder Rakshit Pally
December 2021

Accepted by:
Dr Nina Christine Hubig, Committee Co-Chair
Dr Vidya Samadi, Committee Co-Chair
Dr Amy Apon, Committee Member
ABSTRACT

Floods are among the most destructive natural hazards that affect millions of people across the world leading to severe loss of life and damage to property, critical infrastructure, and the environment. Deep learning algorithms are exceptionally valuable tools for collecting and analyzing the catastrophic readiness and countless actionable flood data. Convolutional neural networks (CNNs) are one form of deep learning algorithms widely used in computer vision which can be used to study flood images and assign learnable weights and biases to various objects in the image. Here, we leveraged and discussed how connected vision systems can be used to embed cameras, image processing, CNNs, and data connectivity capabilities for flood label detection. We built a training database service of >9000 images (image annotation service) including the image geolocation information by streaming relevant images from social media platforms, South Carolina Department of Transportation (SCDOT) 511 traffic cameras, the US geological Survey (USGS) live river cameras, and images downloaded from search engines. All these images were manually annotated to train the different models and detect a total of eight different object categories. We then developed a new python package called “FloodImageClassifier” to classify and detect objects within the collected flood images. “FloodImageClassifier” includes various CNNs architectures such as YOLOv3 (You look only once version 3), Fast R-CNN (Region-based CNN), Mask R-CNN, SSD MobileNet (Single Shot MultiBox Detector MobileNet), and EfficientDet (efficient object detection) to perform both object detection and segmentation simultaneously. Canny edge detection and aspect ratio concepts are also included in the package for flood water level estimation and classification. The pipeline is smartly designed to train a large number of images and calculate flood water levels and inundation areas which can be used to identify flood depth, severity, and risk. “FloodImageClassifier” can be embedded to the USGS live river cameras or 511 traffic cameras to monitor river and road flooding conditions and provide early intelligence to decision makers and emergency response authorities in real-time.
DEDICATION

To my parents, Ashok Kumar Pally; Sudha Vani

To my advisors, Drs. Vidya Samadi and Nina Hubig

To my team at Hydroinformatics and Hydroresources Research (HHR) Lab
at Clemson University
ACKNOWLEDGMENTS

Thanks to my committee: Dr Vidya Samadi, Dr Nina Cristine Hubig, Dr Amy Apon and the HHR team. They spent a large amount of time to help me establish and complete this research work. This work is supported by the U.S. National Science Foundation (NSF) Directorate for Engineering under grant CBET 1901646. Any opinions, findings, and discussions expressed in this study are those of the authors and do not necessarily reflect the views of the NSF. Clemson University is acknowledged for generous allotment of compute time on the Palmetto cluster. The USGS and DOT are acknowledged for providing free of charge live river and road flooding images. The code is publicly available at the HHR FloodImageClassifier repository.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE PAGE</td>
<td>i</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
</tbody>
</table>

## CHAPTER

I. INTRODUCTION ........................................................................................................ 1

II. RESEARCH OBJECTIVES AND RELATED WORK .................................................... 4

   - Research Questions and Motivation ............................................................. 4
   - Related Work ................................................................................................... 4

III. METHODOLOGY ..................................................................................................... 8

   - CNN Classifier ................................................................................................. 8
   - Generic Object Detection ................................................................................ 10
   - Regional-Based Networks for Object Detection ............................................. 11
   - Regression-Based Networks for Object Detection ......................................... 17
   - Removal of Detected Objects and Flood Depth Estimation ......................... 24
   - Performance Metrics ....................................................................................... 27
   - Flood Database System and Data Collection Module .................................... 29
   - FloodImageClassifier System Architecture ................................................ 31

IV. APPLICATION ........................................................................................................ 33

   - Object Detection ............................................................................................... 33
   - Detection and Inpainting .................................................................................. 39
   - Flood Depth and Inundation Area Estimation ................................................ 40

V. FUTURE WORK and CONCLUSION ......................................................................... 44
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Water levels with associated aspect ratios and flood severity and risk estimation</td>
</tr>
<tr>
<td>2</td>
<td>Summarizes the performance of detection task in flood photos</td>
</tr>
<tr>
<td>3.1</td>
<td>Prediction scores of different object detection algorithms on custom flood dataset</td>
</tr>
<tr>
<td>3.2</td>
<td>Prediction scores of different object detection algorithms on MSCOCO dataset</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>The architecture of the CNN flood image classifier developed in this research</td>
</tr>
<tr>
<td>2</td>
<td>The Fast R-CNN network architecture comprising of fully connected layers with Softmax probabilities</td>
</tr>
<tr>
<td>3</td>
<td>Bounding box priors and bounding box predictions</td>
</tr>
<tr>
<td>4</td>
<td>IoU calculation using the ground truth box and the predicted bounding box</td>
</tr>
<tr>
<td>5</td>
<td>The architecture of the “FloodImageClassifier” package</td>
</tr>
<tr>
<td>6</td>
<td>The Fast RCNN detection results with bounding boxes</td>
</tr>
<tr>
<td>7</td>
<td>The Mask RCNN detection results with bounding boxes</td>
</tr>
<tr>
<td>8</td>
<td>YOLOv3 detection results with bounding boxes</td>
</tr>
<tr>
<td>9</td>
<td>SSD MobileNet detection results with bounding boxes</td>
</tr>
<tr>
<td>10</td>
<td>EfficientDet detection results with bounding boxes</td>
</tr>
<tr>
<td>11</td>
<td>Detection and Inpainting Pipeline</td>
</tr>
<tr>
<td>12</td>
<td>Mild water depth estimation</td>
</tr>
<tr>
<td>13</td>
<td>Moderate water depth estimation</td>
</tr>
<tr>
<td>14</td>
<td>Severe water depth estimation</td>
</tr>
</tbody>
</table>
CHAPTER ONE
INTRODUCTION

Floods are on the rise globally with the frequent recorded events occurring during the past few years in the US alone. These extreme events pose a considerable threat to human life and results in destructive damage to property, critical infrastructure, and communities (Philips et al., 2018). During flooding events, citizens around the world increasingly act as human sensors and collect and share millions of flood images and videos on social media to record flood magnitude, damage, and impacts. Multimedia images, videos, geotagged texts posted over social media platforms such as Facebook, Twitter, YouTube, Flickr, and other online forums can provide valuable real-time information about flood situation. By using the content and user metadata from volunteered geographic information shared online, we can identify potential at-risk neighborhoods around the inundation areas that have been flooded. In addition, real time surveillance cameras have been installed by several agencies such as the US Geological Survey (USGS) and the Department of Transportation (DOT) across numerous river and road networks to meet the need for timely assessment of road and river flooding situations (Donratanapat et al., 2020). These real time videos/images can be used to track increasing flood levels during a storm and continuously monitor the potential impacts of flooding on nearby locations. Videos and time lapse images can also be processed to extract image frames and related information, which can be used to measure a range of flood characteristics such as flood depth and inundation areas. Indeed, accurate and efficient assessment of real time images is crucially important to assess road and other critical infrastructure conditions during storm. This
information provides timely and useful details on the hazards to avoid when flooding has occurred. This includes areas to avoid when using a vehicle, and safety and health hazards such as downed electrical lines, flooded roads, etc. (e.g., Chaudhary et al., 2019).

Various methods have recently been proposed to monitor floodwater level and crowd sourced techniques have been recently implemented for flood monitoring and label detection (e.g., Ning, 2019; Chaudhary et al., 2019; Alizadeh Kharazia and Behzadan, 2021). While these studies provided significant insights into the application of crowdsourcing in flood detection and assessment, challenges are still presented by the overwhelming amount of unlabeled, unfiltered images produced through social media streams and extracting potentially useful information to manage data streaming volumes. Computer Vision is the science of understanding and processing digital images and videos that can be used to extract meaningful features and detect flood labels accurately from massive number of crowd sourced images/videos generated and shared during or after the event across various social media platforms. Computer vision has proved to be useful in a number of applications with the advent of deep learning (e.g., Nie et al., 2018; Bantupalli et al., 2018; Brunetti et al., 2018). The use of deep learning in computer vision can be categorized into various categories such as classification of images and videos, segmentation, and detection within images and videos. Object detection is a Computer Vision task that automatically localizes multiple objects into categories of interest from images (e.g., Brunetti et al., 2018; Redmon et al., 2016; Wang et al., 2014; Girshick et al.,
Object detection does not only concentrate on classifying images, but also tries to accurately identify the location of objects contained within the image and label the concepts to get a better understanding of the images.

However, detecting multiple objects that comprise of our visual world within flooded images is a challenging task owing to the large number of variations in object appearances due to pose, illumination conditions, and scaling. The objects are often embedded within scenes in clutter, sometimes alongside with other objects that are previously unseen. There are also a large number of object categories, with each category having a wide variety of appearances. The above-mentioned factors along with a lack of visual experience often fool recognition systems, hence this field has gained much attention in the recent years.

This research is the first attempt known to the authors that used various CNN-based models for flood image labeling, inundation area calculation, and flood level classification. We applied YOLOv3, Fast R-CNN, Faster R-CNN (Ren et al., 2015), Mask R-CNN (He et al., 2017), SSD MobileNet (Liu et al., 2016), and EfficientDet (Tan et al., 2020) for generic object detection, flood label detection and flood depth estimation. Based on the basic CNN architectures, generic object detection and flood label detection were achieved by classification and bounding box regressions whereas, flood surveillance and flood label detection were achieved using pixel-level segmentation techniques. We integrated these
methods with a proposed flood level classification approach (flood severity and risk) to develop a new Python package called “FloodImageClassifier”. This Python package has been tested using a large number of images streamed from various federal agencies web services and social media platforms.

This research is the first attempt known to the authors that used various CNN-based models for flood image labeling, inundation area calculation, and flood level classification. We applied YOLOv3, Fast R-CNN, Faster R-CNN (Ren et al., 2015), Mask R-CNN (He et al., 2017), SSD MobileNet (Liu et al., 2016), and EfficientDet (Tan et al., 2020) for generic object detection, flood label detection and flood depth estimation. Based on the basic CNN architectures, generic object detection and flood label detection were achieved by classification and bounding box regressions whereas, flood surveillance and flood label detection were achieved using pixel-level segmentation techniques. We integrated these methods with a proposed flood level classification approach (flood severity and risk) to develop a new Python package called “FloodImageClassifier”. This Python package has been tested using a large number of images streamed from various federal agencies web services and social media platforms.

This dissertation is organized as follows. In Section 2, related work along with the research questions and motivation of this research work are explained. The proposed methodologies, procedures, algorithms, and the functionality of “FloodImageClassifier”.py are introduced.
and discussed in Section 3. Section 4 discusses the implementation and case study applications. Conclusions and future works and limitation of the tool are provided, respectively in Sections 5 and 6.

CHAPTER TWO
RESEARCH OBJECTIVES AND RELATED WORK

2.1 Research Questions and Motivations

Developing the flood image classifier package involved addressing the following research questions:

1) How to programmatically identify at-risk areas to flooding based on real-time information extracted from crowdsourced images/videos streamed from various web services and social media platforms.

2) How can these real-time videos/images be used to track increasing flood levels during a storm (i.e. estimate the flood depth during flooding situations) and continuously monitor the potential impacts of flooding on nearby locations.

Our aim was to develop an end-to-end pipeline integrated with the Tensorflow Object Detection API and IBM cloud service based on the above-mentioned research questions and estimate, track and monitor the water depth in order to identify the at-risk flooding locations by making use of crowdsourced multimedia data.
2.2 Related Work

Before the fast growth of deep learning techniques for image recognition and classification, bag-of-words (BoW) was one of the most popular technique for image classification. Descriptors such as Scale Invariant Feature Transform (SIFT, 1999) and Speeded Up Robust Features (Bay et al., 2006) are used to extract all the features from the images and form a vocabulary by considering each individual feature as a word. But with the BoW approach, it was hard to keep track of the context and extract various features from images. In recent years, deep neural networks have widely used to perform many images processing tasks. The reason for the popularity of deep learning models is TensorFlow an open-sourced deep learning framework which provides users the access to pre-trained deep learning classification (and regression) models with flexible training on users’s custom dataset.

Significant advancements were recently made with the development of various Convolutional neural networks (CNNs) algorithms such as R-CNN (Region-based CNN). The R-CNNs with deep architectures follow a different approach compared to the shallow learnable architectures that have the capacity to learn complex features and informative object representations without having to design the features manually (LeCun et al., 2015). Since the advent of R-CNNs, advanced object detection models have been designed. This
includes Fast R-CNN (Girshick, 2015) which optimizes classification and bounding box regression tasks by employing an additional sub-network to generate regional features (Ren et al., 2015). Another good example of such models is YOLOv3 (Redmon et al., 2016) model which performs object detection by making use of a fixed-grid regression approach (Redmon et al., 2016). All these models bring about significant performance improvements over the traditional BoW model and make real-time flood object detection a more achievable task.

While the field of image processing for flood label detection is important, there is very few studies that focused on this topic thus far. For example, Yang et al., (2014) implemented visual recognition method to read water levels from a river camera to predict flood depth due to rising water levels. Laplacian method (Vincent and Folorunso, 2009) and probabilistic Hough transform (Zhu & Brilakis, 2009) were utilized in Yang et al., (2014)’s study for detecting the edges of different objects and the straight waterline, respectively. In another study, Pan et al., (2018) computed water level by remotely monitoring the length of a measuring ruler in footage using CNNs. They found that CNNs outperformed other traditional image processing algorithms with a standard deviation of 6.69 mm. Ning, (2019) implemented CNNs to screen flooding photos from social media and detect labels. More recently, Park et al., (2021) estimated flood depth through detecting submerged vehicles in flooded photos using Mask R-CNN (see He et al., 2017) and compared calculated flood depth with the most similar 3D rendered objects based on feature maps extracted by Visual
Geometry Group Nets (VGGNets; Simonyan & Zisserman, 2015). Their proposed approach achieved absolute error values as low as 6.49 cm in flood depth calculation. More recently, Alizadeh Kharazia and Behzadan, (2021) used image processing and deep learning to study flood depth using traffic stop signs as ubiquitous measurement benchmarks in flood photos. In their study, flood depth was estimated with a mean absolute error of 12” in crowdsourced photos.
In this section we discuss about the basic concepts of convolutional neural networks and their applications in various image processing tasks such as image classification and object detection.

3.1 CNN Classifier

Various CNNs were used in this research for the classification task while *keras* (Chollet et al., 2015) deep learning library was used to build the CNNs classifier. The CNN model was trained for 27 epochs with a batch size of 72. The images were partitioned into train and validation sets in the ratio of 9:1. The images present within the training set were resized by scaling the pixels prior to providing the images as input to the CNNs. Our developed CNN is illustrated in Figure 1 that consists of the following layers:

**Input layer:** The first layer of the CNN is the input layer which takes an image as input, resizes the image and passes the image onto the next layer for feature extraction.

**Convolutional Layers:** Three convolutional layers were designed in the model to apply small filters on each part of the image, match the feature points within the image and extract features from the image.
**Pooling Layer:** The extracted features are passed onto the pooling layer, which helps in reducing the special dimensions by shrinking the images down while preserving the most important information within them. It picks the highest values from each region that is retains the best fits of each feature within that region.

**Rectified Linear Unit Layer (ReLU):** This layer normalizes the obtained values by replacing the negative values obtained from the pooling layer with zeros to help the CNN stay mathematically stable.

**Fully Connected Layers:** This is the final layer which takes the filtered images as input and then divides them into categories along with their respective labels and scores.

Figure 1. The architecture of the CNN flood image classifier developed in this research.

Based on the CNN structure presented in Figure 1, we implemented six CNN algorithms in this research. Each algorithm along with its mathematical structure is explained below.
3.2. Generic Object Detection Methods

Generic object detection methods can be classified into two types. One with the traditional object detection pipeline which involves the generation of region proposals and then classifying each of these proposals into different object categories. The models which can be created under regional proposal method are R-CNN (see Girshick, 2015; He and Gkioxari, 2017 for more information), SPP-net (SPP-net modifies RCNN with a Spatial Pyramid Pooling layer; see He et al., 2015), Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al., 2015), R-FCN (region-based fully convolutional networks; Li et al., 2016), feature pyramid networks (FPN; Lin et al., 2017) Mask R-CNN (He and Gkioxari, 2017), and some ensemble methods which are a combination of the above-mentioned models. The second method involves treating object detection as regression or classification problem and make use of a unified architecture to obtain the result directly. The models which fall under this category include MultiBox (making multiple predictions containing boundary boxes and confidence scores; see Erhan et al., 2017), AttentionNet (Yoo et al., 2015), G-CNN (grid-based CNN; Najibi et al., 2016), YOLOv3, SSD (Single Shot Detector; Liu et al., 2016), DSSD (Deconvolutional Single Shot Detector; Fu et al., 2017), and DSOD (Deeply Supervised Object Detectors; Shen et al., 2017).

3.2.1 Regional Based Networks for Object Detection
The regional proposal-based framework follows a two-step process, which involves scanning the entire image first and then identifying, focusing on the regions of interest. To achieve this a CNN is inserted into the sliding window method, which predicts bounding boxes directly from locations of the most important feature map after obtaining the confidence scores of underlying object categories. Each layer in the CNN model is called a feature map, the feature map input layer is nothing but a 3D matrix which consists of pixel intensities for different color channels (e.g., RGB). Different types of transformations can be applied to the feature maps such as filtering and pooling. Filters convolute the filter matrix containing the values of a receptive field of neurons and uses a non-linear function such as sigmoid or ReLU to obtain the final response. There are a variety of pooling operations such as max pooling, average pooling, L2-pooling and local contrast normalization, the objective of pooling operations is to summarize the responses of a receptive field (i.e., set of neurons connected to a small portion of adjacent neurons from the previous layer) into one value to produce more robust feature descriptors. Next, we have the fully connected layers which are used to fine tune the initial feature hierarchy in a supervised manner in order to adapt to different visual tasks. Based on the visual tasks involved, different activation functions are used to build the final layer to get a specific conditional probability for each output neuron. Objective functions such as mean squared error or cross-entropy loss were used to optimize the network via the SGD (Stochastic Gradient Descent) method. The family of models selected for the object detection task are described below:
**Fast R-CNN**: The architecture of the Fast R-CNN processes the whole image using convolutional layers and produces the feature maps. The region of interest (RoI) pooling layers is then used to extract fixed-length feature vectors from each RoI. The generated feature vectors are passed on to the fully connected layers before inputting them to the output layers (see Figure 2). There are two output layers out of which one is responsible for producing SoftMax probabilities for each of the categories and the second output layer is responsible for defining the bounding boxes with four real-valued numbers which represent the edges of the bounding box. All the parameters excluding the generation of regional proposals are optimized using multi-task loss. The multi-task loss function is used to jointly train classification and bounding box regression and is defined by Equation 1:

\[
L(p, u, t^u, v) = L_{cls}(p, u) + \lambda [u \geq 1] L_{loc}(t^u, v)
\]  

(Equation 1)

Where, \(L_{cls}(p, u) = -\log p_u\) calculates the log loss for the classes \(u\) and \(p_u\) based on the probability distribution \(p = (p_0, p_1, \ldots, p_C)\) over the \(C+1\) output generated from the Fully Connected layer. \(L_{loc}(t^u, v)\) is defined over the predicted offsets \(t^u = (t^u_x, t^u_y, t^u_w, t^u_h)\) and ground-truth bounding-box regression targets \(v = (v_x, v_y, v_w, v_h)\), where \(x, y, w,\) and \(h\) denote the two coordinates of the box center, width, and height, respectively. Each \(t_u\) adopts the parameter settings to specify an object proposal with a log-space height/width shift and scale invariant translation. The Iverson bracket indicator function \([u \geq 1]\) is employed to omit all background RoIs. To provide more robustness against outliers and
eliminate the sensitivity in exploding gradients, a smooth $L_1$ loss was adopted to fit bounding box regressors using Equations (2) and (3).

$$L_{loc}(t^u, v) = \sum_i \epsilon_{x,y,w,h} \text{smooth } (t^u_i - V_i)$$  \hspace{1cm} \text{(Equation 2)}

Where, $\text{smooth}_{L1}(x) = (0.5 x^2 \text{ if } |x| < 1; |x| - 0.5 \text{ otherwise})$  \hspace{1cm} \text{(Equation 3)}

To further accelerate the pipeline Fast R-CNN samples the mini-batches in a hierarchical manner where $N$ images are sampled randomly. First RoIs are sampled in each image. Here $R$ is the number of regions of interest. Also, RoIs from the same image share the computational power and memory in the forward and backward passes. In the Fast R-CNN, all networks’ layers are trained in a single step using a multi-task loss which significantly improves the accuracy and efficiency of object detection task.

Figure 2. The Fast R-CNN network architecture comprising of fully connected layers with Softmax probabilities.
**Faster R-CNN:** Ren et al., (2015) introduced an additional Region Proposal Network (RPN) so called Faster R-CNN, which significantly reduces the computational burden by sharing full-image convolutional features with detection network. The RPN architecture includes a fully-convolutional network, which simultaneously predicts the object bounds and the scores at each stage, creates rectangular object proposals for each of the identified objects. RPN operates on a specific convolutional layer with the preceding layers shared with object detection network. The network slides over the entire convolutional feature map with the help of $n \times n$ spatial window. Each sliding window produces a low-dimensional vector which is passed on to the two fully connected layers which are box-classification layer and box-regression layer. In other words, the architecture is composed of an $n \times n$ convolutional layer network followed by two sibling $1 \times 1$ convolutional layers with ReLU applied to the output of an $n \times n$ convolutional network. The bounding box-regressions are achieved by comparing proposals relative to reference boxes (i.e., anchors). In the Faster R-CNN anchors of 3 different scales and aspect-ratios are used. The loss function is defined as follows:

$$L(p_i, t_i) = (1/ N_{cls}) \sum_i L_{cls} (p_i, p_i^*) + \lambda (1/ N_{reg}) \sum_i p_i^* L_{reg} (t_i, t_i^*)$$  \hspace{1cm} \text{(Equation 4)}$$

Where $p_i$ shows the predicted probability of the $i$-th anchor being an object. The ground truth label $p^* i$ is 1 if the anchor is positive, otherwise 0. $t_i$ stores 4 parameterized coordinates of the predicted bounding box while $t^* i$ is related to the ground truth box overlapping with a positive anchor. $L_{cls}$ is a binary log loss and $L_{reg}$ is a smoothed $L_1$ loss.
These two terms are normalized with the mini-batch size \((N_{cls})\) and the number of anchor locations \((N_{reg})\), respectively. In the form of fully-convolutional networks, Faster R-CNN can be trained end-to-end by back-propagation and SGD in an alternate training manner. With the introduction of Faster R-CNN, it was possible to train regional proposal-based architectures in an end-to-end manner. However, there were some other disadvantages such as time-consuming training and RPN shortcoming in generating object instances instead of object-like regions to deal with those objects with complex shapes.

**Mask R-CNN:** Mask R-CNN is a simple extension of the Fast R-CNN with a class label and a bounding-box offset. We included a third branch for predicting the segmentation masks for each object instance. Mask R-CNN object masks are different from the class and bounding-box outputs produced by Fast R-CNN. The Mask R-CNN adopts a two-stage approach. The first stage consists of a regional proposal network and in the second stage along with predicting the class labels and the bounding-box offsets, Mask R-CNN also provides a binary mask for each RoI. This approach does not align with most of the existing classification systems where classification of objects depends on the mask predictions. In this way it is similar to the Fast R-CNN approach which performs bounding-box classification and regression together.

The loss of the entire multi-task approach for each RoI is defined as

\[
L = L_{\text{class}} + L_{\text{box}} + L_{\text{mask}},
\]

where the classification loss and bounding-box loss are the same as defined in Freund and Schapire (1997). The mask branch has \(K m \times m\) binary masks, one for each class. To
design this algorithm, we added a per-pixel sigmoid and defined the $L_{\text{mask}}$ which is the average binary cross-entropy loss. Due to this definition of $L_{\text{mask}}$, the network generates masks for each class without competition among other classes. This separates the mask and class prediction tasks. Specifically, we predicted an $m \times m$ mask from each RoI using an FCN. This allowed each layer in the mask branch to maintain an explicit $m \times m$ object spatial layout without collapsing it into a vector representation that lacks spatial dimensions. This kind of fully convolutional arrangement requires fewer parameters and at the same time is more accurate when compared with the other models. Certain amount of misalignment is introduced between the RoI and features due to the coarse spatial quantization performed by the RoI pooling function in the Faster R-CNN. The Mask R-CNN solves this problem using a quantization-free layer called RoIAlign, which helps in preserving the per-pixel spatial correspondence. The RoIAlign layer replaces the RoI pooling quantization layer with a bilinear interpolation function as discussed by Jaderberg et al., (2015) which computes the values of input features at the four regularly sampled locations in each RoI bin. These changes help Mask R-CNN to significantly improve the accuracy and precision of object detection tasks. Given that Mask R-CNN only adds a small amount of computational overhead to the Faster R-CNN architecture, this makes it very simple to implement with great instance segmentation and object detection results. In other words, Mask R-CNN is an accurate and flexible instance detection model which can be used for a wide range of images.

3.2.2 Regression-based Networks for Object Detection
The region-based networks consist of several correlated layers, which include generating regional proposals, feature extraction, classification and bounding box regressions. On the other hand, regression/classification-based frameworks directly perform mapping from bounding box coordinates and class probabilities, thereby greatly reducing the time spent to complete the task. Two significant frameworks that make use of regression/classification-based networks for object detection are the SSD and YOLOv3.

**SSD-Mobilenet:** The SSD-Mobilenet is a combination of SSD network and CNN MobileNet. SSD-MobileNet is a kind of regression model, which makes use of the features from various convolutional layers to build classification regression and bounding box regression. Each feature map consists of $k$ frames that contrast in size and width-to-height ratio. These frames are called as default boxes, the default boxes are then scaled to form feature maps that can be calculated as:

$$S_K = S_{\text{min}} + \frac{S_{\text{max}} - S_{\text{min}}}{(m-1)} \sum_{(kE[1, m])} (k-1), (kE[1, m]) \quad \text{(Equation 5)}$$

The $m$ denotes the total number of feature maps and $S_{\text{min}}$ and $S_{\text{max}}$ are parameters that need to be set while configuring the training job. Loss function is calculated as the sum of the confidence loss $L_{\text{conf}}(s, c)$ of the classification regression and the position loss $L_{\text{loc}}(r, l, g)$ of the bounding box regression. The function can be depicted as:

$$L(s, r, c, l, g) = \frac{1}{N} (L_{\text{conf}}(s, c) + \alpha L_{\text{loc}}(r, l, g)) \quad \text{(Equation 6)}$$
Where $\alpha$ is a constraint to manipulate the confidence loss and position loss; $s$ and $r$ are the eigenvectors representing the confidence loss and position loss respectively; $c$ is the confidence of classification; $l$ is the offset of predicted box which includes both translations offset and scaling offset; $g$ is the alignment box (ground-truth box) of the objective genuine position; and $N$ is the quantity of default boxes that coordinate the alignment boxes of this classification. SSD-MobileNet is consists of point wise layers and depth wise layers. The depth wise layers are deep convolutional layers that utilizing $3 \times 3$ kernel while point wise layers are common convolutional layers utilizing $1 \times 1$ kernel. Batch normalization and activation function used to rectify linear unit 6 (ReLU6) that are applied on every convolutional result.

**EfficientDet-D1:** EfficientDet-D1 is a neural network architecture and one of the Tensorflow object detection API. EfficientDet-D1 runs faster than other detectors largely follow the one-stage detectors paradigm just like SSD and YOLOv3 (YOLO version 3). The EfficientDet-D1 architecture is split into two parts, the first part is the backbone network which consists of pretrained EfficientNets and the second part is a BiFPN feature network which takes features from the backbone network and continuously applies top-down and bottom-up feature fusion. Next, a box network takes these fused features as input and produces the bounding boxes and class predictions, respectively.

EfficientDet-D1 had multiple state-of-the-art model variants, ranging from D0-D7 with D0 being the lightweight model, therefore requires less compute resources and D7 being the
heavy weight model that requires more computational support. EfficientDet-D1 is faster than other detectors and it uniformly scales the resolution, depth, and width. We linearly increased the BiFPN depth and the BiFPN width was exponentially increased, as well. Basically, a grip search is performed, and the best value is selected as the BiFPN width scaling factor. The depth and width are scaled according to the following Equations:

\[
W_{BiFPN} = 64 \cdot (1.35 \phi); \quad D_{BiFPN} = 3 + \phi \quad \text{ (Equation 7)}
\]

Where, \( \phi \) is the compound coefficient which controls the scaling dimensions. For the prediction layers the width is same as the BiFPN network, but the depth is linearly increased using the following equation:

\[
D_{Box} = D_{class} = 3 + \left\lfloor \frac{\phi}{3} \right\rfloor \quad \text{ (Equation 8)}
\]

Where \( D_{Box} \) and \( D_{class} \) represent the box/class prediction network. BiFPN, box/class net, and input size are scaled up using Equations 7 and 8, respectively.

**YOLOv3**: Redmon et al. (2016) proposed a novel object detection framework called YOLOv3, which makes use of the whole topmost feature map to predict both confidences for multiple categories and bounding boxes. YOLOv3 is an algorithm that directly predicts the class probabilities and bounding box offsets by applying a single feed forward neural network (originally a version of GoogLeNet, later updated and called DarkNet based on...
VGG) for the entire image. YOLOv3 is one of the faster object detection algorithms, it considers object detection as a regression problem and eliminates region proposal generation and feature resampling by encapsulating all stages in a single network to form a true end-to-end detection system. The algorithm splits the input image into small grid cells and each cell predicts a bounding box and the class label. The result is a large number of candidates bounding boxes that are consolidated into a final prediction by a post-processing step. We used a pre-trained YOLOv3 model to perform object detection on unseen flood images. We defined a keras model that had the right number and type of layers to match the pre-trained model weights. The model predicted multiple candidates bounding boxes referring to the same objects. The list of bounding boxes can be filtered and those boxes that overlap and refer to the same object can be merged by defining and passing the amount of overlap as a configuration parameter. This filtering of bounding box regions is generally referred to as non-maximal suppression and is a required post-processing step. The number of boxes were further reduced by retrieving only those that strongly predict the presence of an object. This is achieved by enumerating over all boxes and checking the class prediction values.

As shown in the Figure 3, YOLOv3 mainly consists of two things, a feature extraction layer called Darknet-53 and the YOLOv3 convolutional layers. The convolutional layers output five basic parameters of the detection result (b_x, b_y, b_w, b_h, and confidence), where, b_x and b_y are coordinates representing the center of the object label and b_w, b_h are the width and height of the object label and the confidence is the prediction score for that particular
object label. Along with the prediction probability YOLOv3 also weights the bounding boxes. YOLOv3 makes use of the same core idea used by the Faster R-CNN for region extraction, the only change is $k$-means clustering that is used for determining the bounding box priors. Compared to classification-based systems, YOLOv3 has several advantages such as (i) it learns more context information as it passes the entire image as input to the detection system, (ii) YOLOv3 is much faster when compared to the R-CNN algorithm and it is about 1000 times faster that Faster R-CNN and 100 times faster that Fast R-CNN. The feature extraction layer Darknet-53 is the same as mentioned above in the YOLOv3 algorithm. The detection accuracy of YOLOv3 is similar to that of SSD, but it is three times faster than SSD. Object detection using YOLOv3 consists of four steps:

1) Predicting bounding boxes: YOLOv3 anchor boxes are obtained by clustering. As shown in Figure 3, an input image is divided into $s \times s$ grid cells and for each bounding box YOLOv3 predicts the four coordinate values $(t_x, t_y, t_w, t_h)$, for the predicted cell, the width, and the height of bounding box prior $p_w, p_h$ are computed based on the coordinates of the upper left corner of the image $(c_x, c_y)$. 
The YOLOv3 algorithm makes use of logistic regression to predict the bounding boxes which is different from the approach used by the Fast R-CNN. Each ground truth object is assigned only one bounding box prior and if a bounding box prior is not assigned then it does not incur any loss due to coordinates or class predictions. YOLOv3 eliminates the extraction of feature frames from each region and instead it divides the image into $S \times S$ grids, and the size of the a priori frame is set to the size of the object frame of the $k$-means clustering dataset.

2) Class prediction: Multilabel classification is used to predict the classes that may be contained within the bounding box. YOLOv3 does not make use of a softmax layer but instead it employs an independent logistic classifier. Binary cross-entropy loss is then used during training for class prediction.

3) Prediction across scales: Similar to the Fast R-CNN, YOLOv3 predicts object at three different scales. In our experiment three different boxes were predicted for
each scale. In addition, YOLOv3 makes use of a multi-scale detector. This connects the feature maps from three different scales of $52 \times 52$, $26 \times 26$, $13 \times 13$ from the feature extraction layer for the detection and regression of the object by the detector. The connection of the deep feature map is beneficial to the learning of the large object feature information, and the connection of the shallow feature map is more conducive to the learning of the small target feature information.

4) Feature extractor: The YOLOv3 uses Darknet-53 for feature extraction instead of the VGG16 network used by Fast R-CNN. The Darknet-53 comprises of $3 \times 3$ and $3 \times 1$ convolutional layers and hopping connection layers to perform feature extraction, which is more inclined towards learning of feature information of the previous item.

In this research, we developed our own keras YOLOv3 model and then used it to make predictions on unseen flooding images. We designed the keras model (i.e., the number and type of layers used) based on the pretrained model weights. These weights were obtained by training the Darknet-53 code based on the Microsoft COCO (MSCOCO) dataset (Lin et al., 2014). Next, we loaded the model weights using the weight reader class which was saved within the working directory. In order to make predictions we loaded the input images and pre-processed the images before feeding them to the model. The pre-processing involved converting the images into a square shape of size $416 \times 416$. To load and resize the image we converted the PIL (Pillow python library) image object into a NumPy array.
and then rescale the pixels from 0-255 to 0-1 floating point values. Once the image is pre-processed, we fed the images to the model.

The model predicted a huge number of bounding boxes, we then filtered these bounding boxes regions using a method known as non-maximal suppression which merges bounding boxes that have a certain amount of overlap and are referring to the same object. This reduces the bounding boxes considerably, leaving very few boxes of interest. Next, these bounding boxes were rescaled to the original shape, size and drawn around each detected object. Finally, the model generated a plot of the original images with the bounding boxes drawn for the detected objects along with the class labels for the objects and their respective prediction scores.

3.3. Removal of Detected Objects and Flood Depth Estimation

The results generated by each of the above-mentioned state-of-the art object detection models are fed into an object removal system. The object removal system helps in removing of detected object and reconstructing the image in a plausible manner by using exemplar-based inpainting method (Criminisi et al., 2003). Image inpainting involves filling in the voids within an image, this is used in a number of applications such as reconstruction of old images and damaged videos, removal of unwanted image content such as superimposed text, and removal of objects. This method uses partial convolutions with an automatic mask update to achieve state-of-the-art results. It substitutes convolutional layers with partial convolutions and mask updates, as a result the links are not skipped in a U-Net (convolutional networks for fast and precise segmentation of
images) and making it possible to achieve good inpainting results. This algorithm uses a combination of texture synthesis and inpainting methods to identify the target region which needs to be filled in and a source region which is used as a reference to fill in the target regions. Our final object removal pipeline takes an image as input, detects the location of various objects within the image and produces an image with the removed detected objects as the final output. This task is performed in order to detect the edges of the water surface using Canny Edge Detection (Zhao et al., 2017) as it calculates the surface areas of water which in turn are used to determine the floodwater level. Canny Edge Detection (Canny, 1986) is a popular multi-stage edge detection algorithm explained step by step below:

**Noise Reduction:** Edge detection is susceptible to noise, therefore in the first step the algorithm tries to smooth the noise using a gaussian filter.

**Finding Intensity Gradient of the Image:** A Sobel operator is used to filter the smoothened image in order to obtain a derivative both in the horizontal ($G_x$) as well as vertical ($G_y$) directions. With the help of these images, it is possible to identify the edge gradient and the directions for each pixel as shown below:

\[
\text{Edge Gradient} (G) = \sqrt{G_x^2 + G_y^2} \quad \text{ (Equation 9)}
\]

\[
\text{Angle(}\theta\text{)} = \tan^{-1}(G_y/G_x) \quad \text{ (Equation 10)}
\]
Gradient direction is always perpendicular to the edges. It is mostly approximated to one of the four different angles that represent the horizontal, vertical, and two diagonal directions.

**Non-maximum Suppression:** After obtaining the gradient direction and magnitudes, the entire image is scanned to identify and remove unwanted pixels that do not contribute towards the edge (i.e., the point is checked to see if it forms a local maximum with the neighboring points, in that case it is considered for the next stage, otherwise, it is suppressed). This is done by checking each pixel and determining if it is a local maximum in its neighborhood in the direction of the gradient.

**Hysteresis Thresholding:** In this step we discuss which edges should be taken into consideration (i.e., which ones are the real edges, and which are not). To do so, we used two threshold values, i.e., edges with intensity gradient values greater than the maxVal (sure edges) and edges with intensity gradient values below minVal (non-edges). The edges whose gradient intensity lies between the maxVal and minVal are classified as edges or non-edges based on their connectivity. If these edges are connected to a sure-edge then they are also considered to be a part of the edge and if not they are considered as non-edges and discarded. The output from the detection and inpainting pipeline is then used to estimate the water depth. We identified the surface water edges and calculated the water depth using the Canny Edge Detection algorithm, the aspect ratio concept (Xu-kai et al., 2012), and OpenCV package which includes powerful functions to handle computer vision
tasks such as smoothening and thresholding. First, we detected the edges of the water surface, drew contours around the water surface and then calculated the area of the contours (i.e., the area of the water surface). Next, based on the aspect ratio which is calculated by taking into consideration the area of the water surface detected within the image, we estimated the water levels. We categorized the water levels into mild (1.26-1.8), moderate (0.54-1.25), and severe (0.18-0.53) conditions to reflect flood severity and risk.

Table 1. Water levels with associated aspect ratios and flood severity and risk estimation.

<table>
<thead>
<tr>
<th>Water Level</th>
<th>Aspect Ratio</th>
<th>Flood Severity and Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>&gt;1.8</td>
<td>Mild</td>
</tr>
<tr>
<td>Level 2</td>
<td>1.62 – 1.8</td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td>1.44 – 1.62</td>
<td></td>
</tr>
<tr>
<td>Level 4</td>
<td>1.26 – 1.44</td>
<td></td>
</tr>
<tr>
<td>Level 5</td>
<td>1.08 – 1.25</td>
<td>Moderate</td>
</tr>
<tr>
<td>Level 6</td>
<td>0.90 – 1.08</td>
<td></td>
</tr>
<tr>
<td>Level 7</td>
<td>0.72 – 0.90</td>
<td></td>
</tr>
<tr>
<td>Level 8</td>
<td>0.54 – 0.72</td>
<td></td>
</tr>
<tr>
<td>Level 9</td>
<td>0.36 – 0.53</td>
<td>Severe</td>
</tr>
<tr>
<td>Level 10</td>
<td>0.18 – 0.36</td>
<td></td>
</tr>
<tr>
<td>Level 11</td>
<td>&lt;0.18</td>
<td></td>
</tr>
</tbody>
</table>

3.4. Performance Metrics

Flood image object detection can be challenging since both the probability of occurrence of a particular object and the position of the object should be precisely predicted. Hence, standard metrics such as accuracy and precision that are widely used for evaluating image classification models cannot be used for examining flood object detection models. We used Mean Average Precision (MAP) which is a popular performance metric to evaluate algorithms that involve predicting the object location as well as classifying the probability of occurrence. MAP evaluates the correctness of bounding box prediction using a metric
called Intersection over Union (IoU). IoU is a ratio between the intersection and the union of the predicted boxes, and the ground truth boxes. This metric is also called as the Jaccard Index since it was first published by Paul Jaccard in the early 1900s.

![Figure 4. IoU calculation using the ground truth box and the predicted bounding box.](image)

True positive (TP) and true negative (TN) in Equations 11 and 12 refer to the number of correct detections; TP represents positive (correct) detections while TN indicates negative (incorrect) detections. False positive (FP) and false negative (FN), on the other hand, refer to the number of incorrect detections; FP denotes positive detections while FN represents negative (incorrect) detections (Powers, 2020). Next, average precision (AP) will be calculated by plotting precision as a function of recall and calculating the area under the curve.
\[ Precision = \frac{TP}{(TP+FP)} \quad \text{(Equation 11)} \]
\[ Recall = \frac{TP}{(TP+FN)} \quad \text{(Equation 12)} \]

### 3.5. Flood Database System and Data Collection Modules

In this research, we built for the first time a flood dataset consisting of > 9000 flooding images collected from various sources. The primary data sources are Twitter, US department of Transportation (DOT) 511 traffic cams, the US Geological Survey (USGS) river cameras, YouTube, and search engines videos. Extracting and downloading images from Twitter is a tedious and time-consuming task since only about 10% of the tweets have images attached to them (Francalanci et al., 2017). Images can be programmatically collected in two ways i.e., Twitter Representational state transfer Application Programming Interface (REST API) or Streaming API. The Rest API enables user to collect a list of tweets with images, user id, etc., whereas the Streaming API allows users to collect tweets with images in real-time based on search terms, user ids or locations. Streaming real-time tweets is relatively simple but has some downsides, as well. For example, using the Twitter API we can only access tweets from the past 7 days. We used a group of keywords such as “floods”, “flood emergency”, “disaster risk”, flooded roads”, etc., to collect Twitter images and to further filter the queries geocode i.e., the latitude and longitude values of the images that were passed to the API in order to stream geolocation information.
To collect real-time Twitter images, we used Streaming API through the tweepy python package to download real-time tweets in JSON format that contained an URL for the corresponding images. To collect flooding images for the US, we used the “country_code” attribute present in geotagged tweets JSON to filter the tweets accordingly. Overall, >1000 tweets were collected which were filtered further based on the inclusion of certain features within the images such as houses, cars, and trees. The presence of these features in flooded images is critical in identifying appropriate images with geolocation information for label detection and floodwater classification. Utilizing search engines such as Google and Bing and existing flooding image datasets from other resources included additional >4800 flooding images to enrich the dataset. Specifically, a sizeable number of videos were collected using feeds from YouTube, DOT traffic surveillance cameras, the USGS river cameras, and other online sources. Although, the images are collected from different sources, they can form a preliminary training dataset for the CNNs. These live footages are readily available and can be exploited to obtain valuable, qualitative flooding information.

In this study, frames (i.e., flooding images) are extracted from these footages using the OpenCV package. These live surveillance footages are then broken frame by frame and after every 10 seconds a frame labeled with the camera location and time stamp is pushed into the image data store.

3.6. “FloodImageClassifier” System Architecture

“FloodImageClassifier” workflow is illustrated in Figure 5. Different workflows of the “FloodImageClassifier” package are described including (i) the data collection module, (ii)
classification of images using a trained classifier, and (iii) object/label detection using object detection models. In this tool, the position of the object in a flooded image is defined by rectangular coordinates. The pipeline of object detection models is primarily divided into three phases: (i) informative region selection, (ii) feature extraction, and (iii) classification. In a flooded image, different objects may appear at different locations of the image and have different sizes and aspect ratios, so it is important to scan the entire image using a sliding window and select the regions of interest. This would reflect the information region selection by object detection models. The flood image classification, the object label detection, and floodwater level classification are three main components of the “FloodImageClassifier” package. These three components are embedded within the same system architecture but are trained separately and follow different workflows. The large training set built using the data collection module helps in improving the performance of the classifier as well as the object detector. To verify the extendibility of the system, a pre-trained YOLOv3 model was also implemented using Keras and TensorFlow that is included to the list of custom trained object detection models. The test images go through the classifier as well as the object detector (i.e., CNNs) and the inferences drawn of labels, flood levels, and flood risk classification are displayed to the user.
Figure 5. The architecture of the “FloodImageClassifier” package. The package can also be implemented through Flood Analytics Information System (FAIS) application (see Donratanapat et al., 2020).
CHAPTER FOUR
APPLICATION

We tested the aforementioned state-of-the-art approaches on flood datasets which include a custom dataset (collected flood images) built by collecting images from various sources as explained in section 2.1 as well as the MSCOCO dataset that contains flood and non-flood images. This study used Mask R-CNN, SSD MobileNet, YOLOv3, Fast R-CNN and EfficientDet to compute flood inundation area and depth using these images. The data were split into training and test portions. The test images were then passed to the object detection models to draw inferences and outputs. This process helped detect objects within the images, create bounding boxes for the identified objects along with their class labels and respective prediction scores.

4.1. Object Detection

We split the image dataset into training and test sets and annotated the images with our defined custom object categories. We considered several object categories such as vehicle, forest, tree, traffic sign, water vessels, residential areas, and bridges. Annotation of images involved highlighting each of the objects within an image manually using bounding boxes and labeling them appropriately. An image annotation tool that supports YOLOv3 format (called LabelImg) was used to annotate the images. Before passing these annotated images as input to the object detection models for training, the .xml annotation files are converted to Tensorflow record files and passed as input for training the models along with the images. Next, we trained the convolutional based CNN for image classification. Pretrained
models from the Tensorflow model zoo were used for object detection that are trained on larger datasets such as ImageNet (Russakovsky et al., 2015), MSCOCO dataset (Lin et al., 2014). After the training is completed, we extracted the newly trained custom object detection inference graphs, exported, and saved them in a separate folder within the same directory. These saved models were later used to perform object detection (i.e., perform inferences).

By default, the training process logs some of the basic performance metrics which along with the test images were used to evaluate the trained models and performance metrics. To view these basic performance measures, we performed an evaluation process on the trained models which used the checkpoint files (i.e., snapshot of the models at given steps) generated during the training process and evaluated how well the model detects objects in the test dataset. These basic evaluation metrics generated during the evaluation process can also be used to compute the MAP for the models.

The test dataset was consisted of many challenging images with most of the images having more than one object. We used MAP which is a popular performance metric to evaluate algorithms that involve predicting the object location as well as classifying the probability of occurrence. We also used IoU to determine whether the detection was correct or not. It was also observed that as the training dataset size was increased the MAP value also improved. This allowed us to further increase the IoU threshold from 0.5 to 0.75. To evaluate the performance of object detection algorithms, the IoUs are detected, and ground-
truth masks was calculated by dividing the overlapping area between the two masks by their union area. We first computed the true positives, false positives, true negatives, and false negatives. IoU was then used to get the true positives and false positives, that is whether the detection was correct or not was determined by comparing the IoU with a threshold value. Generally, 0.5 is used as the threshold value and if the IoU is greater than 0.5 we consider it as a true positive, otherwise it is considered as a false positive. Since the ground truth data already provided the information about the actual number of objects within the image, we then calculated the true negatives (i.e., part of the image where the object was not predicted) and false negatives (i.e., the objects that were missed out by our model). Using these values and IoU thresholds, we calculated a number of correct detections for each class in an image indicating the precision and recall values (Equations 11 and 12). Next, we chose different confidence thresholds ranging from 0.3 to 0.7. From a total of >9000 dataset images, 7000 were included since some images depict partially visible the flooded areas. Table 2 summarizes the performance of detection task in flood photos.

Table 2 summarizes the performance of detection task in flood photos.

<table>
<thead>
<tr>
<th>IOU (%)</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>AP (%)</th>
<th>Processing Time (s)</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>69.9</td>
<td>79</td>
<td>63.7</td>
<td>1.55</td>
<td>Mask RCNN</td>
</tr>
<tr>
<td>75</td>
<td>71</td>
<td>76</td>
<td>62.9</td>
<td>2.52</td>
<td>Fast RCNN</td>
</tr>
<tr>
<td>75</td>
<td>61</td>
<td>59.7</td>
<td>48</td>
<td>1.22</td>
<td>SSD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MobileNet</td>
</tr>
<tr>
<td>75</td>
<td>65.4</td>
<td>63</td>
<td>58</td>
<td>1.21</td>
<td>EfficientDet</td>
</tr>
</tbody>
</table>
Next, we used exported inference graph (i.e., saved custom object detection models) to perform inference on some external flood images that were not a part of our collected flood image dataset to understand how well each of these models detect the various object categories on which they were trained. As illustrated in Figures 6-10, it is evident that the custom trained models were capable of detecting multiple objects within a single image as they almost detected 90% of the objects precisely. However, object detection models produced different outcomes. Segmentation models such as Mask R-CNN identified the foreground shapes and highlighted the objects using bounding boxes and drawing a mask on the object. This helped in clearly segmenting one object from another one whereas other object detection models such as Fast R-CNN, YOLOv3, EfficientDet, and SSD MobileNet highlighted the detected objects using only a single bounding box.

Figure 6. The Fast RCNN detection results with bounding boxes.
Figure 7. The Mask RCNN detection results with bounding boxes.

![Input Image](image1) ![Detection Result](image2)

Figure 8. YOLOv3 detection results with bounding boxes.

![Input Image](image3) ![Detection Result](image4)

Figure 9. SSD MobileNet detection results with bounding boxes.

![Input Image](image5) ![Detection Result](image6)

Figure 10. EfficientDet detection results with bounding boxes.

The prediction scores of different models were calculated for different object categories namely vehicle, person, forest, tree, traffic sign, residential area (i.e., houses), water vessels (i.e., boats, ships, etc.) and bridges/dams by passing the same set of test images to each of
these models (Table 3). This determined which of these models is the best object detection approach for flood images.

Table 3.1 The prediction score of different object detection models using collected flood dataset.

<table>
<thead>
<tr>
<th>Models/Object Categories</th>
<th>Vehicle</th>
<th>Forest</th>
<th>Traffic Sign</th>
<th>Tree</th>
<th>Residential Area</th>
<th>Person</th>
<th>Water Vessels</th>
<th>Critical Infrastructure (bridge, dam, road, storm water facilities, railroad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD MobileNet</td>
<td>92%</td>
<td>54%</td>
<td>60%</td>
<td>97%</td>
<td>51%</td>
<td>74%</td>
<td>98%</td>
<td>95%</td>
</tr>
<tr>
<td>Fast R-CNN</td>
<td>99%</td>
<td>56%</td>
<td>99%</td>
<td>89%</td>
<td>100%</td>
<td>99%</td>
<td>48%</td>
<td>100%</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>85%</td>
<td>70%</td>
<td>87%</td>
<td>73%</td>
<td>96%</td>
<td>91%</td>
<td>67%</td>
<td>89%</td>
</tr>
<tr>
<td>YOLOv3</td>
<td>99.9%</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>98.3%</td>
<td>95%</td>
<td>N/A</td>
</tr>
<tr>
<td>EfficientDet</td>
<td>62%</td>
<td>35%</td>
<td>42%</td>
<td>51%</td>
<td>69%</td>
<td>59%</td>
<td>43%</td>
<td>59%</td>
</tr>
</tbody>
</table>

We also tested MSCOCO dataset that consists of 300,000 fully segmented images and each image on an average includes about 7 object instances from a total of 80 different object categories. Object detection results using MSCOCO is shown in Table 3.2.

Table 3.2 The prediction score of different object detection models using MSCOCO dataset.

<table>
<thead>
<tr>
<th>Models/Object Categories</th>
<th>vehicle</th>
<th>Forest</th>
<th>Traffic Sign</th>
<th>Tree</th>
<th>Residential area</th>
<th>Person</th>
<th>water vessels</th>
<th>Critical Infrastructure (bridge, dam, road, storm water facilities, railroad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD MobileNet</td>
<td>70%</td>
<td>N/A</td>
<td>36%</td>
<td>N/A</td>
<td>N/A</td>
<td>41%</td>
<td>30%</td>
<td>N/A</td>
</tr>
<tr>
<td>Fast R-CNN</td>
<td>99%</td>
<td>N/A</td>
<td>95.9%</td>
<td>N/A</td>
<td>N/A</td>
<td>98</td>
<td>45%</td>
<td>N/A</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>98%</td>
<td>N/A</td>
<td>94%</td>
<td>N/A</td>
<td>N/A</td>
<td>99%</td>
<td>99%</td>
<td>N/A</td>
</tr>
<tr>
<td>Model</td>
<td>Detection Score</td>
<td>Confidence</td>
<td>Model</td>
<td>Detection Score</td>
<td>Confidence</td>
<td>Model</td>
<td>Detection Score</td>
<td>Confidence</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>------------</td>
<td>-------</td>
<td>-----------------</td>
<td>------------</td>
<td>-------</td>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>YOLOv3</td>
<td>99.4%</td>
<td>N/A</td>
<td>99.96%</td>
<td>N/A</td>
<td>96.9%</td>
<td>N/A</td>
<td>94.9%</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Each of the above models pretrained on the MSCOCO dataset were downloaded from the Tensorflow model zoo along with their corresponding label files for this dataset. Next, we performed the inference on the same set of test images which were used for the custom trained models. Table 4 represents the detection scores obtained by running inference on each of the models and it was observed that the models trained on the COCO dataset were unable to detect certain object labels such as trees, residential areas (i.e., houses) and critical infrastructures such as bridges, dams, etc. On the other hand, each of these models when trained on the custom dataset (our collected flood images) were capable of detecting the above-mentioned object labels accurately. Several visual features in flooded photos led to misdetection or highly erroneous label detection. The top four sources of errors include image background, darkness in surface water, water reflection, and wavy surface water.

### 4.2. Object Removal and Inpainting

Once the detection results were generated by each of these models, we attempted to remove each of the detected objects and reconstructed the image by filling the void spaces in a plausible manner using exemplar based inpainting (see Criminisi et al., 2003). Image inpainting involves filling in the voids within an image, this is used in a number of applications such as reconstruction of old images and damaged videos, removal of unwanted image content such as superimposed text and removal of objects. We used partial convolutions with an automatic mask update to achieve state-of-the-art results. Image
inpainting model substituted convolutional layers with partial convolutions and masked the updates. This algorithm successfully identified the target region which was filled using the surrounding areas of the target region as reference.

Figure 11. Detection and inpainting pipeline.

4.3. Flood Depth and Inundation Area Estimation

The final output of the object detection and image inpainting pipeline was used to estimate the water depth using canny edge detection and aspect ratio concept as discussed in the methodology section. Given an input image, first we resized the image and converted it into a grayscale. Next, we identified and eliminated the skyline because both water and skyline had the same color gradient, and it was possible that the skyline could also be detected as a water surface. Once the skyline was eliminated, only a portion of images consisting of the water surface was taken into consideration. As shown in Figure 11, we
detected the edges of the water surface (i.e., draw the contours) and then calculated the area of the water surface that was printed on the original image. Having identified the water surface correctly, a bounding box was then drawn around the contour and its aspect ratio was calculated (i.e., the ratio between the width and the height of the water surface). This aspect ratio value was then used to identify the water level and was printed on the original image (see Table 5).

Figure 12. Mild water depth estimation (level 4) using canny edge detection and aspect ratio approach along with calculated area which was 55073.6 pixels.

Figure 13. Moderate water depth estimation using canny edge detection and aspect ratio approach along with calculated area which was 42621 pixels.
Figure 14. Sever water depth estimation using canny edge detection and aspect ratio approach along with calculated area which was 77308 pixels.

As shown in Figures 12-14, the image is resized and converted into a grayscale image before processing it. The image is firstly smoothened using a gaussian kernel prior to the thresholding operation. This helped remove the noise while keeping the underlying structure of the image intact. Next, we manually defined a threshold value of 9 in our case and performed a thresholding operation using OpenCV’s `threshold()` function. The result of the thresholding operation is a binary image which nicely captures the water surface along with a few other objects such as people, cars, and trees with the help of masks. To isolate the water surface from the other objects that were highlighted, we used the OpenCV’s `findContours()` function to identify the foreground mask shapes and draw contours around them. Next, we calculated the area of each of the contours, sorted the contour areas and only the largest contour was printed over the original image, allowing us to clearly segment the water surface. After highlighting the water surface using contours, a bounding box was drawn around the contour to calculate the aspect ratio. The aspect ratio value is used to determine the water level as shown in Figures 12-14. If the aspect ratio is in a range of 1.26-18, the water level is then considered to be low (mild), if the aspect ratio is in a range of 0.54-1.26, the water level is considered to be moderate, and if the aspect...
ratio is in a range of 0.18-0.54, the water level is considered to be high (sever flood risk). Based on these classification, Figure 12 showed a floodwater level 4 with 55703.6 pixels area, and mild flood risk and severity while Figures 13 and 14 revealed moderate and severe flood risk conditions with flood levels 6 and 11, respectively. These results reveled the important of canny edge detection and aspect ratio concept for flood severity classification. However, due to the variation in appearances, lighting conditions and backgrounds, it is difficult to manually design a robust feature descriptor to perfectly describe and classify all kinds of objects, floodwater levels, and the inundation areas in a flooded image.
CHAPTER FIVE
CONCLUSION AND FUTURE WORK

This research examined various CNNs algorithms for flood label detection. We used YOLOv3, Fast R-CNN, Mask R-CNN, SSD MobileNet, and EfficientDet algorithms to label flood objects, classify flood levels, and estimate inundation areas. A training dataset of >9000 flooding images was built by streaming relevant images from social media platforms and various online sources and these photos were used to train the CNN architectures for flood label detection. Once all the images were collected, labelImg was used to annotate these images with eight different object categories namely boat, bridge, car, house, person, traffic signs, trees, and trucks. Next, these images were split along with their respective .xml files into train and test sets in a 9:1 ratio i.e., 90% for training and 10% for testing. We first employed pretrained Tensorflow object detection models trained on the MSCOCO dataset and pretrained YOLOv3 as label detection and instance segmentation model. We trained YOLOv3 on two sets of data, i.e., our own collected flood data and MSCOCO dataset and then segment the centroid-pixel annotated mitosis ground truths and produced the mitosis mask and bounding box labels. Among different algorithms used in this study, Mask RCNN showed promising results. A major advantage of the Mask R-CNN approach is the ability to perform both detection and instance segmentation of surface water within the images, this allowed the development of further algorithms to perform tasks such as water level classification and inundation area calculation. Specifically, the ability of Mask R-CNN for flood depth detection is crucial for real-time
monitoring of water level rising that can provide early warning to decision makers and emergency response authorities. Flood instance segmentation presented in the paper is the first step towards developing a real-time image-based flood monitoring system for at-risk environments.

The Mask R-CNN detection performance was found to be affected by the IoU threshold where the higher threshold led to multiple predicted (flooded) regions within a single image and the lower threshold resulted in lacking predicted flooded region. To evaluate the detection performance quantitatively and select the optimal threshold, the F1 scores and precision metrics were assessed over different thresholds. It was observed that when the size of training dataset increased the MAP value also improved. This allowed us to further increase the IoU threshold from 0.5 to 0.75. The results indicated that the threshold at 0.75 performs better with average precision of more than 89%. Although, this optimal threshold may vary application by application (see Papandreou et al., 2017; Schneider et al., 2018) and could be properly adjusted depending on the types of images and labels.

Regardless of the rapid advancements and promising progress in the object detection domain, there are still several challenges that could be addressed in future works. One important challenge is the detection of small objects in the images. CNN-based detection algorithms have a number of combinations of convolutions followed by a pooling layer. This allows the network to resize the images from ~600×600 resolution down to ~30×30. Due to this fact small object features extracted on the first layers may disappear somewhere
in the middle of the network and never actually count for detection and classification steps. This is also the reason why CNNs struggle with detecting small objects. Capturing high resolution images, using focal loss in the process of training a neural network (Lin et al., 2020), and employing Feature Pyramid Networks (FPN; see Lin et al., 2017) may improve small object detection in the images. Another important key finding to be considered while modifying the network architecture is scaling. Similar kind of objects present within images usually exist in different scales which is apparent in pedestrian detection. To make the model robust to such scale changes, the model could be trained on multi-scale or scale-invariant detectors. The scale-invariant detectors could make use of powerful backbone architectures such as negative-sample mining and sub-category modeling. For multi-scale detectors, General Adversarial Networks (GAN; Kong et al., 2017) which typically narrows down the representational differences between small object and large objects with a low-cost architecture is an effective way to improve the detection accuracy of small objects.

The second challenge to be addressed is the computational burden. It is a tedious and time-consuming task to manually draw the bounding boxes for each object present within the image. To address this, unsupervised object detection (Croitoru et al., 2017), multiple instance learning (Wang et al., 2014) and object prediction using deep neural networks can be integrated into an existing network architecture and use image-level supervision to detect objects and assign appropriate class labels to them. Another approach is to use pre-trained models (Li et al., 2017) with an optimal network architecture. The third challenge
is the extension of 2D object detection methods for object detection within videos. There is a noticeable drop in the network detection accuracy due to the corrupt object appearances caused by the motion blur and changes of focus in videos. To resolve this, several techniques such as spatiotemporal tubelets (Kang et al., 2017) and Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997) can be employed to establish objects associations between consecutive frames.

From flood monitoring and management perspective, this paper demonstrates the development of a key software component which could lead to annotation capable of autonomously identifying and quantifying flood levels and inundation areas. Our research shows promising steps towards application of computer vision in real time image-based flood monitoring system. These approaches can be embedded to the USGS live river cameras or 511 traffic cameras to monitor river and road flooding condition in real time. Future work can concentrate on assessing Mask R-CNN performance over classification of flood risk and severity more quantitively to further explore the importance of video footage in real time flood monitoring.
REFERENCES


[41] Ning, H., 2019. Prototyping a Social Media Flooding Photo Screening System Based on Deep Learning and Crowdsourcing.


