Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft

Application of image processing and convolutional neural networks for flood image classification and semantic segmentation

R.J. Pally [a], S. Samadi [b,*]

[a] School of Computing, Clemson University, Clemson, SC, USA
[b] Department of Agricultural Sciences, Clemson University, Clemson, SC, USA

ARTICLE INFO

Keywords:
Image processing
Convolutional neural networks
Flood label detection
Floodwater level and extend estimation
“FloodImageClassifier” package

ABSTRACT

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 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, Department of Transportation (DOT) 511 traffic cameras, the US Geological Survey (USGS) live river cameras, and images downloaded from search engines. 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 with the USGS live river cameras and 511 traffic cameras to monitor river and road flooding conditions and provide early intelligence to emergency response authorities in real-time.

1. 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 (Phillips 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) across numerous river networks to meet the need for timely assessment of flood situational awareness (Donratanapat et al., 2020). These real time videos/images can be used to track increasing water 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.

Various methods have recently been proposed to monitor floodwater level and crowd sourced images have been recently implemented for

* Corresponding author.
E-mail address: samadi@clemson.edu (S. Samadi).

https://doi.org/10.1016/j.envsoft.2021.105285
Received 18 November 2021; Accepted 8 December 2021
Available online 10 December 2021

flood monitoring and label detection (e.g., Ning, 2019; Chaudhary et al., 2019; Kharazi and Behzadan, 2021). While these studies provided significant insights into the application of crowdsourced images in flood detection and assessment, challenges are still presented by the overwhelming amount of unlabeled, unfiltered data produces 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 and Xie, 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., 2014). In addition to classifying images, object detection 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.

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 (Sivic and Zisserman, 2003). Descriptors such as Scale Invariant Feature Transform (SIFT; Lowe, 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 own dataset (or custom dataset).

Significant advancements were recently made with the development of various Convolutional Neural Networks (CNNs) algorithms such as R–CNN (Region-based CNN) features. 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 which optimizes classification and bounding box regression tasks (Girshick, 2015; Chemelil, 2021), by employing an additional sub-network to generate regional features (Ren et al., 2015). Another good example of such models is YOLOv3 (You Only Look Once version 3; 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 monitor water levels using a river camera to estimate flood depth due to rising water levels. Other approaches such as Laplacian method (see Vincent and Folorunso, 2009) and probabilistic Hough transform (Zhu et al., 2009) were utilized in Yang et al. (2014) for different objects edge detection and the straight waterline calculation. In another study, Pan et al. (2018) computed flood level remotely by reading 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 3D rendered objects using feature maps that are extracted by Visual Geometry Group Nets (VGGNets; Simonyan and Zisserman, 2014). Their proposed approach achieved absolute error values as low as 6.49 cm in flood depth calculation. 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.

This paper 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 (Girshick, 2015), Mask R–CNN, SSD MobileNet (Single Shot MultiBox Detector MobileNet; Liu et al., 2016a,b), and EfficientDet (efficient object detection; Tan et al., 2020a,b) 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”. “FloodImageClassifier” has been tested for many images by streaming videos/images from various data providers and social media platforms.

This paper is organized as follows. In Section 2,the procedures, algorithms, and the functionality of “FloodImageClassifier”.py are introduced and discussed. Section 3 discusses the results of “FloodImageClassifier”.py implementation. Conclusions and future works are provided in Section 4.

2. Methodology

2.1. 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, 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; Ning et al., 2020). 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, or 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. Technically, a sizeable number of videos were collected using feeds from YouTube, DOT traffic surveillance cameras, the USGS river cameras, and other online sources. These footages and images formed a preliminary training dataset for the CNN algorithms, although, they were collected from different sources. In this study, frames (i.e., flooding images) were extracted from these footages using the "*OpenCV*" PythonPython package. These live surveillance footages were then broken frame by frame and after every 10 s a frame was labeled with the camera location and time stamp was pushed into the image data store.

2.2. 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 Fig. 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.

Based on this structure presented in Fig. 1, we implemented six CNN algorithms in this research. Each algorithm along with its mathematical structure is explained below.

2.2.1. *Generic object detection*

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 et al., 2017 for more information), SPP-net (SPP-net modifies RCNN with a Spatial Pyramid Pooling layer; He et al., 2017), Fast R–CNN (Girshick, 2015), R–FCN (region-based fully convolutional networks; Liu et al., 2016a,b), feature pyramid networks (FPN; Lin et al., 2017a,b) Mask R–CNN (He et al., 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 (see Erhan et al., 2017), Attention Net (Yoo et al., 2015), G-CNN (grid-based CNN; Najibi et al., 2016), YOLOv3, SSD (Single Shot Detector; Liu et al., 2016a,b), DSSD (Deconvolutional Single Shot Detector; Fu et al., 2017), and DSOD (Deeply Supervised Object Detectors; Shen et al., 2017).

2.2.1.1. 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 and 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 (e.g., Yiatrou et al., 2016). Each layer in the CNN model is called a feature map, the feature map input layer is 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 and to get a

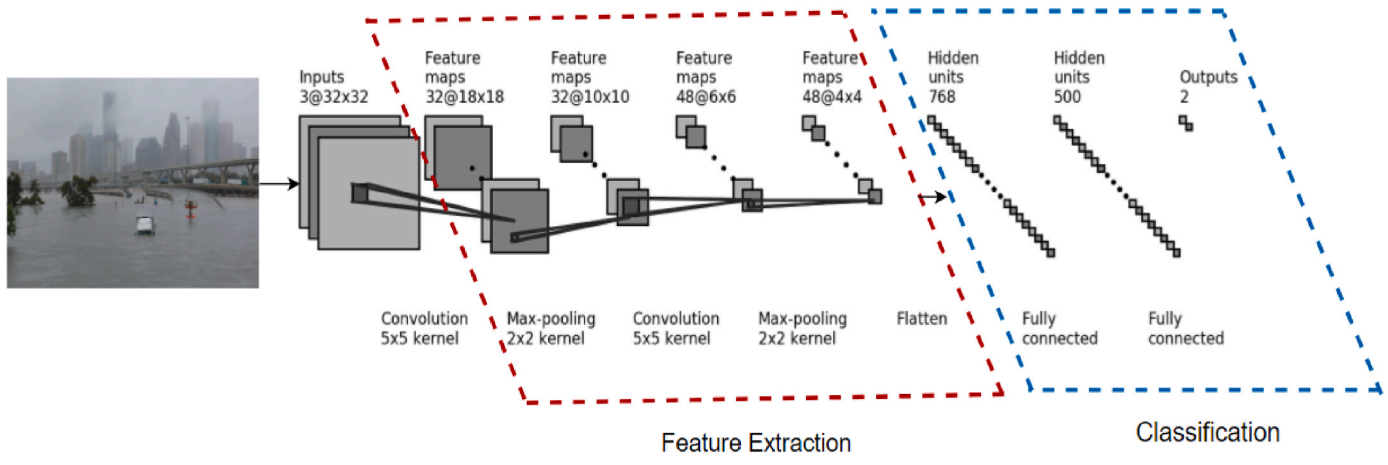

Fig. 1. The architecture of the CNN flood image classifier developed in this research.

specific conditional probability for each output neuron (see Zhao et al., 2019). Objective functions such as mean squared error or cross-entropy loss are 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 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 Fig. 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 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 that is defined by Equation (1):

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

Where, $L_{cls}(p, u) = -\log p_u$ calculates the log loss for the classes u and p_u based on the probability distribution $p = (p_0, p_1, \dots \dots \dots \dots, p_c)$ over the $C+1$ output generated from the FC layer. $Lloc(t\,u,\ v)$ 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 (see Zhao et al., 2019). 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 (Zhao et al., 2019) using Equations (2) and (3).

$$L_{loc}\ (t^u,\ v) = \sum\nolimits_{i\ \in\ x,y,w,h} smooth\ (t_i{}^u - V_i) \tag{2}$$

$$Where,\ smooth_{L1}\ (x) = (0.5\ x^2\ if\ |x| < 1;\ |x| - 0.5\ otherwise \tag{3}$$

To further accelerate the pipeline detection speed, we programmed the Fast R–CNN to sample the mini-batches in a hierarchical manner where N images were sampled randomly. Also, the RoIs from the same image share the computational power and memory in the forward and backward passes (Clark, 2019). In the Fast R–CNN, all networks' layers were trained in a single step using a multi-task loss which significantly improved the accuracy and efficiency of object detection task.

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. The second stage provides a binary mask for each RoI along with predicting the class labels and the bounding-box offsets (see He et al., 2020). 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 $L = L_{class} + L_{box} + L_{mask}$ where the classification loss and bounding-box loss are defined as 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_{mask} which is the average binary cross-entropy loss. Due to this definition of L_{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 (see He et al., 2020). 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 (e.g., Zhao et al., 2019) due to the coarse spatial quantization performed by the RoI pooling function. 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 (also see Zhao et al., 2019). These changes help Mask R–CNN to significantly improve the accuracy and precision of object detection tasks. Mask R–CNN is an accurate and flexible instance detection model which can be used for a wide range of images.

2.2.1.2. *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

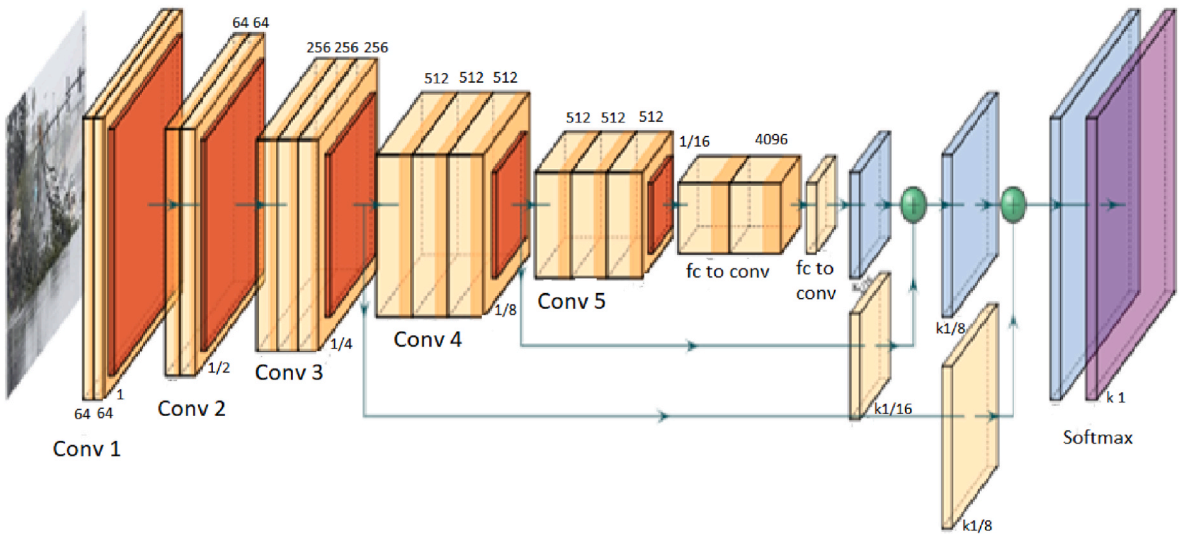

Fig. 2. Fast R–CNN network architecture comprising of fully connected layers with Softmax probabilities.

regressions (see Bouchakwa et al., 2020). 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 SSD and YOLO.

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 (Ali et al., 2019). 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:

$$S_K = S_{min} + \frac{S_{max} - S_{min}}{(m-1)}\left(k-1\right),\ \left(kE\left[1,\ m\right]\right) \tag{4}$$

The m denotes the total number of feature maps and $S_{min,}$ and S_{max} are parameters that need to be set while configuring the training job. Loss function is calculated as the sum of the confidence loss $L_{conf}\,(s, c)$ of the classification regression and the position loss $L_{loc}(r, l, g)$ of the bounding box regression (Ali et al., 2019). The function can be depicted as:

$$L(s,\ r,\ c,\ l,\ g) = \frac{1}{N}\left(L_{conf}(s,\ c) + \alpha L_{loc}(r,\ l,\ g)\right) \tag{5}$$

Where α 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 (Ali et al., 2019). SSD-MobileNet is consists of point wise layers and depth wise layers. The depth wise layers are deep convolutional layers that utilizing 3×3 kernel while point wise layers are common convolutional layers utilizing 1×1 kernel (see Ali et al., 2019). 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 the 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 has 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 (Tan et al., 2020a,b). The depth and width are scaled according to the following Equations:

$$W_{BiFPN} = 64\ .\ (1.35\ \varphi);\ DBiFPN = 3{+}\varphi \tag{6}$$

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

$$DBox = Dclass = 3 + \lfloor \varphi/3 \rfloor \tag{7}$$

Where DBox and Dclass represent the box and class prediction network, respectively. BiFPN, box and class net, and input size are scaled up using Equations (7) and (8), respectively (see Tan et al., 2020a,b for more information). EfficientDet-D1 consistently achieves better accuracy and efficiency than other models from the *TensorFlow* object detection API's model zoo.

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 (Zhao et al., 2019). 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; Brownlee, 2019) 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 (Brownlee, 2019). The model predicted multiple candidates bounding boxes referring to the same objects. The list of bounding boxes was filtered and those boxes that overlapped and referred to the same object were merged by defining and passing the amount of overlap as a configuration parameter (see Brownlee, 2019 for more information). The number of boxes was further reduced by retrieving only those that strongly predicted the presence of an object.

As shown in Fig. 3, YOLOv3 mainly consists of two things, a feature extraction layer called Darknet-53 (Yang et al., 2020) and the YOLOv3 convolutional layers. The convolutional layers output five basic parameters of the detection result (*bx*, *by*, *bw*, *bh*, and *confidence*). Where, *bx* and *by* are coordinates representing the center of the object label and *bw*, *bh* are the width and height of the object label and the *confidence* is the prediction score for that particular object label (Yang et al., 2020). Along with the prediction probability, YOLOv3 also weights the bounding boxes. In the YOLOv3 model *k*-means clustering 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 100 times faster that Fast R–CNN. The feature

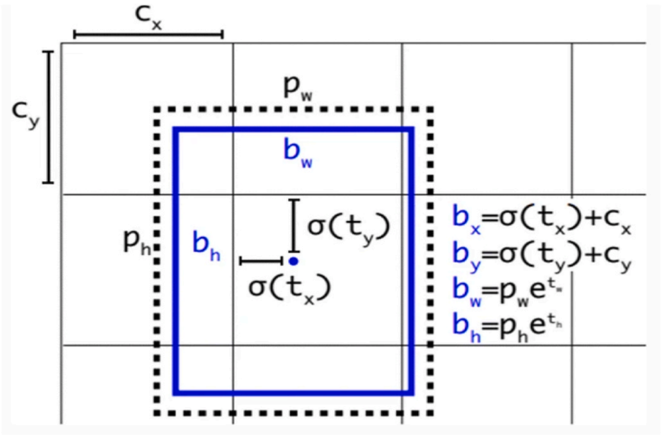

Fig. 3. Bounding box priors and bounding box predictions.

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 (Yang et al., 2020). Object detection using YOLOv3 consists of four steps:

1) Predicting bounding boxes: YOLOv3 anchor boxes are obtained by clustering (Liu et al., 2018). As shown in Fig. 3, an input image is divided into $s \times s$ grid cells and for each bounding box, YOLOv3 predicts the four coordinate values (tx, ty, tw, th), for the predicted cell, the width, and the height of bounding box prior pw, ph are computed based on the coordinates of the upper left corner of the image (cx, cy; See Yang et al., 2020).

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 lose 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 (Yang et al., 2020).

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 52×52, 26×26, and 13×13 from the feature extraction layer for the detection and regression of the object by the detector (e.g., Yang et al., 2020). 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 (Yang et al., 2020).
4) Feature extractor: The YOLOv3 uses Darknet-53 for feature extraction instead of the VGG16 network used by the Fast R–CNN. The Darknet-53 comprises of 3×3 and 3×1 convolutional layers and hopping connection layers to perform feature extraction, which is more inclined towards learning of feature information of the previous item (see Yang et al., 2020 for more information).

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) 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 416×416 sizes. 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 to 255 to 0–1 floating point values (Brownlee, 2019). 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/or refer to the same object. This approach reduced the bounding boxes considerably and left 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.

2.3. Removal of detected objects and flood depth estimation

The results generated by each of the above -mentioned state-of-the art object detection models were fed into an object removal system. The object removal system helped in removing of detected objects and reconstructing the image in a plausible manner by using exemplar-based inpainting method (see Criminisi et al., 2003 for more information). 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) thereby, making it possible to achieve appropriate inpainting results. The object removal system 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 detected objects that can be removed 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., 2014) as it calculates the surface areas of water which in turn are used to determine the water 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:

$$Edge\_Gradient\ (G) = \sqrt{G2x + G2y} \tag{8}$$

$$Angle(\theta) = tan\text{-}1\ (Gy/Gx) \tag{9}$$

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. 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 decide which edges should be taken into consideration (i.e., those that are the real edges vs. those that are non-real ones). 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 the "*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 then categorized the water levels into mild, moderate, and severe conditions to reflect flood severity and risk.

2.4. 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 (see Fig. 4; Rezatofighi et al., 2019). This metrics is also called as the Jaccard Index since it was first published by Paul Jaccard in the early 1900s.

True positive (*TP*) and true negative (*TN*) in Equations (10) and (11) refer to the number of correct detections; *TP* represents positive (correct) detections while TN indicates negative (incorrect) detections (e.g., Kharazi and Behzadan, 2021). 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 (Kharazi and Behzadan, 2021).

$$Precision = TP\,/\,(TP + FP) \tag{10}$$

$$Recall = TP\,/(TP + FN) \tag{11}$$

2.5. “FloodImageClassifier” system architecture

“FloodImageClassifier” workflow is illustrated in Fig. 5. It visually describes different workflows of the “FloodImageClassifier” package including (i) 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 that may 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 were embedded within the same system architecture but were trained separately to follow different workflows. The large training set built using the data collection module helped 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 were 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.

3. Applications

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 and MSCOCO that contain flood and non-flood images. This study used Mask R–CNN, SSD MobileNet, YOLOv3, Fast R–CNN, and EfficientDet algorithms that were pretrained and integrated into “FloodImageClassifier” package. Test images were then passed to these object detection models to draw inferences and outputs. This process helped detect object within the images, create bounding boxes for the identified objects along with their class labels and respective prediction scores.

3.1. 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 were converted to the *TensorFlow* record files and passed as input for training the models along with the images. Next, we trained the convolutional based CNNs for image classification. Pretrained models from the *TensorFlow* model zoo were used for object detection that were trained on larger datasets such as ImageNet (Russakovsky et al., 2015) and MSCOCO dataset (Lin et al., 2014). After the training was 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 a given step) generated during the training process and evaluated how well the model detected the objects in the test dataset. These basic evaluation metrics that were generated during the evaluation process were used to compute the MAP value for each model.

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 increased the MAP value also improved considerably. 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 were first detected, and the ground-truth masks were then calculated by dividing the overlapping area between the two masks by their union area (see Kharazi and Behzadan, 2021). 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 the number of correct detections for each class in an image indicating the

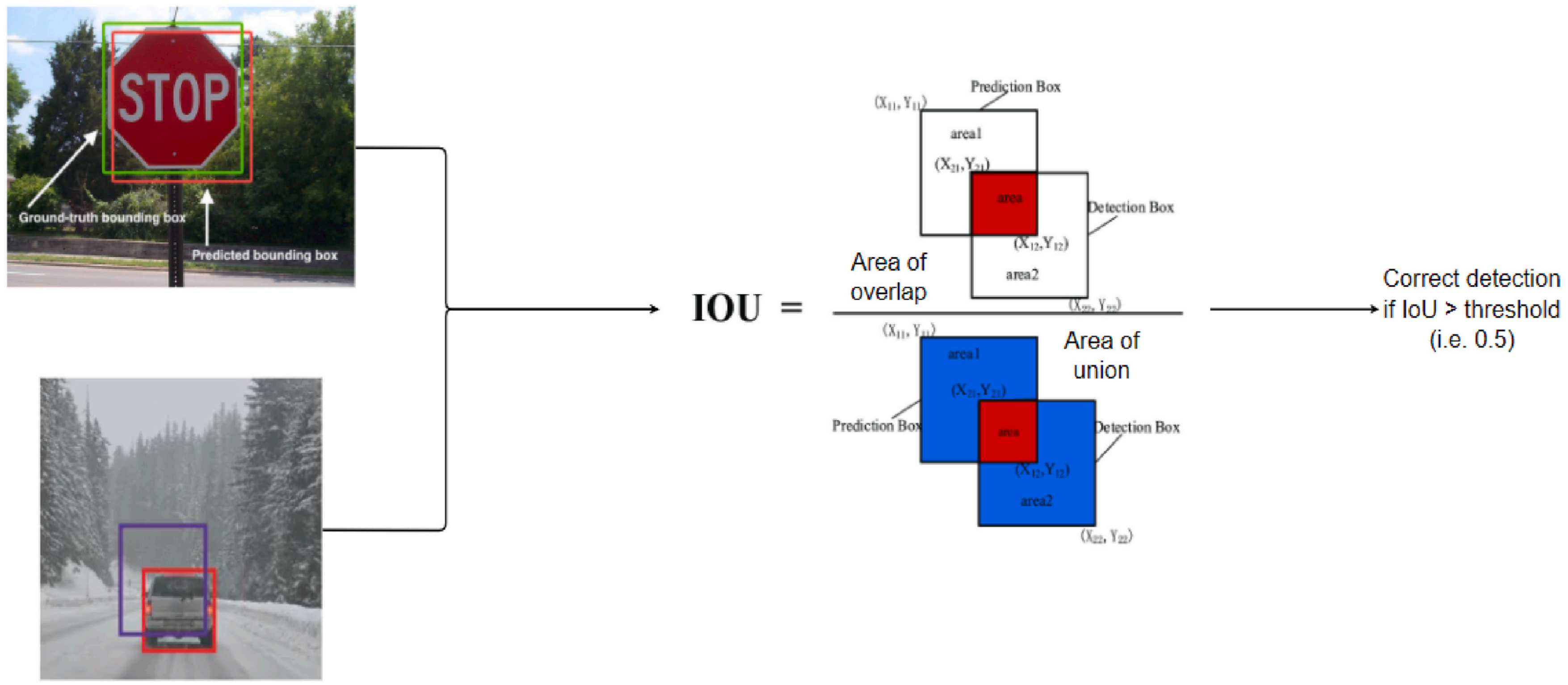

Fig. 4. IoU calculation using the ground truth box and the predicted bounding box.

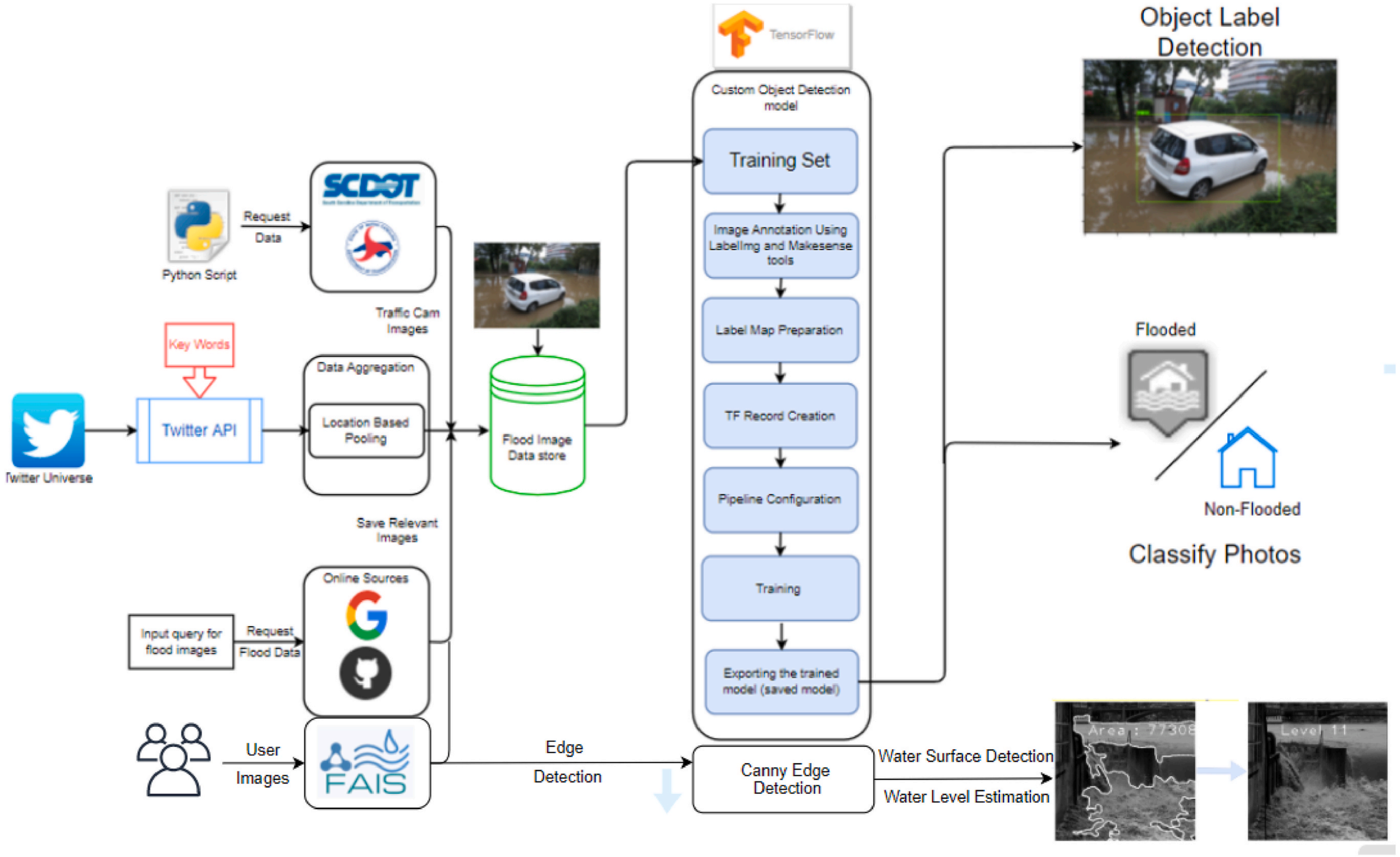

Fig. 5. The architecture of the "FloodImageClassifier" package.

precision and recall values (Equations (11) and (12); see Table 1). Next, we chose different confidence thresholds ranging from 0.3 to 0.7. From a total of >9000 images, > 4000 were included in this analysis since some images depicted partial visibility of the flooded areas.

Next, we used exported inference graph (i.e., saved custom object detection models) to perform inference on some external flood images that were not 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 Figs. 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 by drawing a mask on the object. This helped in clearly segmenting one object from another one whereas other object detection models such as the Fast R–CNN, YOLOv3, EfficientDet, and SSD MobileNet highlighted the detected objects using only a single bounding box.

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 approach determined which of these models was the best object detection algorithm for flood image labeling and detection.

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 are shown in Table 4.

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 object categories in flooded photos led to highly erroneous or misdetection. The top four sources of errors include image background, darkness in surface water, water reflection, and wavy surface water.

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. 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.

Table 1
The performance of each detection model using our customized flood images.

Models	IoU (%)	Precision (%)	Recall (%)	AP (%)	Processing Time (s)
Mask RCNN	75	69.9	79	63.7	1.55
Fast RCNN	75	71	76	62.9	2.52
SSD MobileNet	75	61	59.7	48	1.22
EfficientDet	75	65.4	63	58	1.21

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 Fig. 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 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).

As shown in Figs. 12–14, the image was resized and converted into a grayscale image before processing it. The image was firstly smoothened using a gaussian kernel prior to the thresholding operation. This helped remove the noise while kept the underlying structure of the image intact. Next, we manually defined a threshold value of 9 and performed a thresholding operation using OpenCV's *threshold()* function. The result of the thresholding operation was a binary image which well captured 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 contour, then sorted the contour areas and only the largest contour was printed over the original image. This allowed us to clearly segment the water surface. After highlighting the water surface using contours, a bounding box was illustrated around the contour to calculate the aspect ratio. The aspect ratio value was used to determine the water level as shown in Figs. 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, Fig. 12 showed a floodwater level 4, 55703.6 pixels area with mild flood risk and severity while Figs. 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 label descriptor to completely describe and classify all types of objects, floodwater levels, and the inundation areas in a flooded image.

4. 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 flooded images was first built by streaming relevant images from social media platforms and various data providers and online sources. These photos were then 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 pre-defined object categories namely vehicle, person, forest, tree, traffic sign, residential area (i.e., houses), water vessels (i.e., boats, ships, etc.) and critical infrastructure (bridges, dams, etc.). Next, these images were split into train and test sets in a 9:1 ratio i.e., 90% for training and 10% for testing along with their respective.xml files. We first employed pretrained *TensorFlow* object detection models trained on the MSCOCO dataset and pretrained YOLOv3 as label detection and instance segmentation model. We first trained YOLOv3 on two sets of data, i.e., our customized flood data and MSCOCO dataset and then

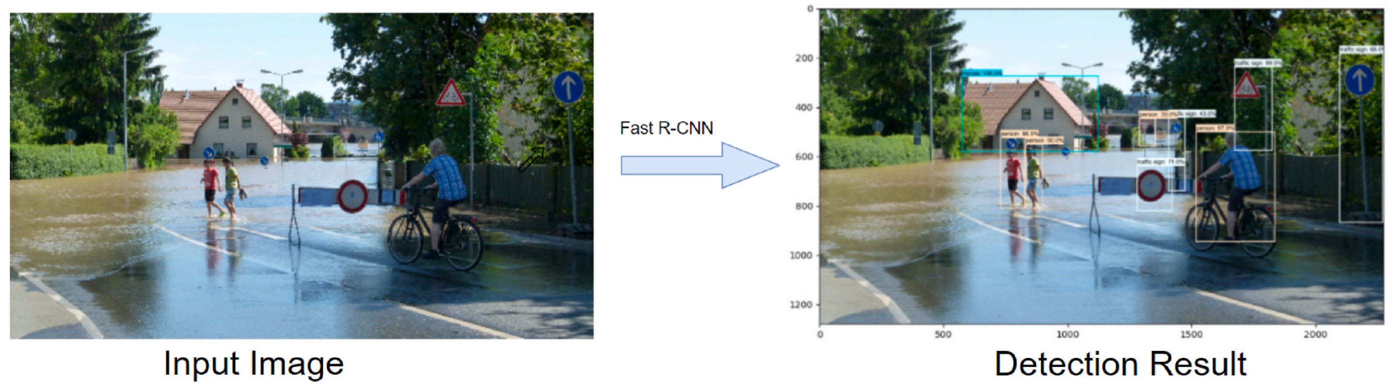

Fig. 6. The Fast RCNN detection results with bounding boxes.

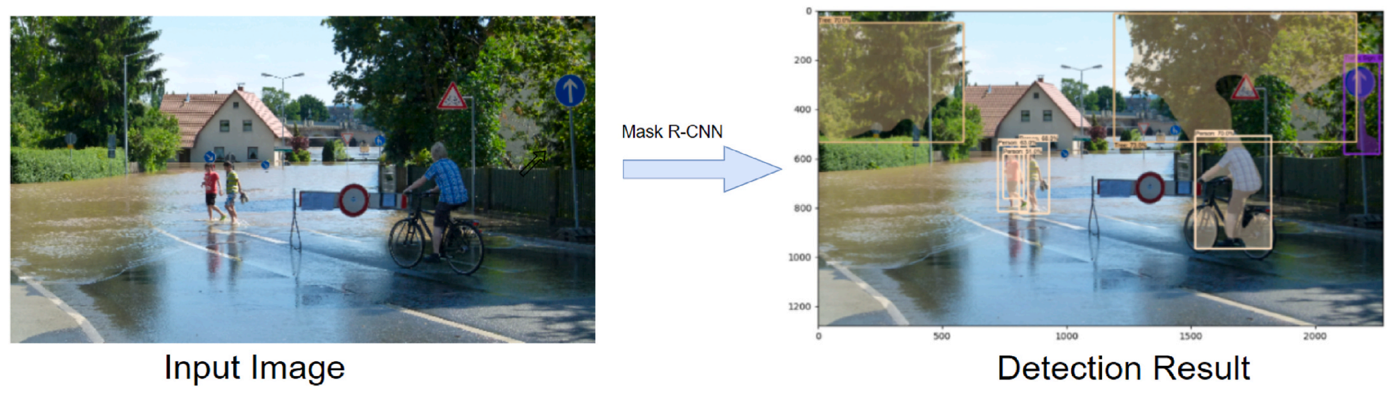

Fig. 7. The Mask RCNN detection results with bounding boxes.

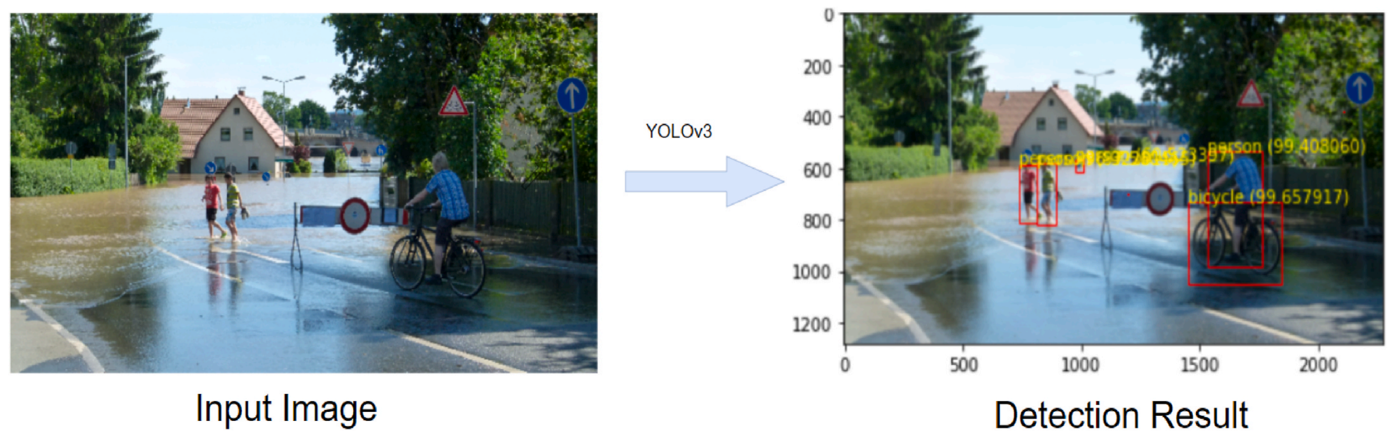

Fig. 8. YOLOv3 detection results with bounding boxes.

segment the centroid-pixel annotated mitosis ground truths and produced the mitosis mask and the bounding box labels. Among different algorithms used in this study, Mask RCNN showed promising results. The Mask R–CNN was able to perform both detection and instance segmentation of surface water within the images. This allowed us to use the MASK R-CNN results to perform other tasks such as water level classification and inundation area calculation. Specifically, the ability of Mask R–CNN for flood depth detection was found to be crucial for real-time monitoring of water level rising that can provide early intelligence to decision makers and emergency response authorities.

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 (also reported in Xu et al., 2020). It was observed that as 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 superior with the precision of >89%. Although, this optimal threshold may vary application by application (see Papandreou et al., 2017; Schneider et al., 2018) while could be properly adjusted depending on the types of images and labels (Xu et al., 2020).

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.

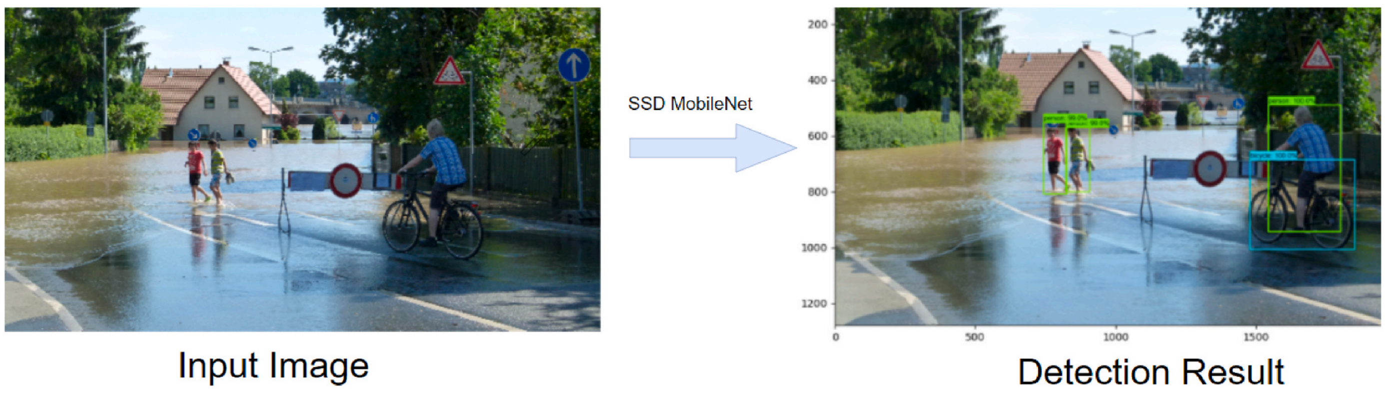

Fig. 9. SSD MobileNet detection results with bounding boxes.

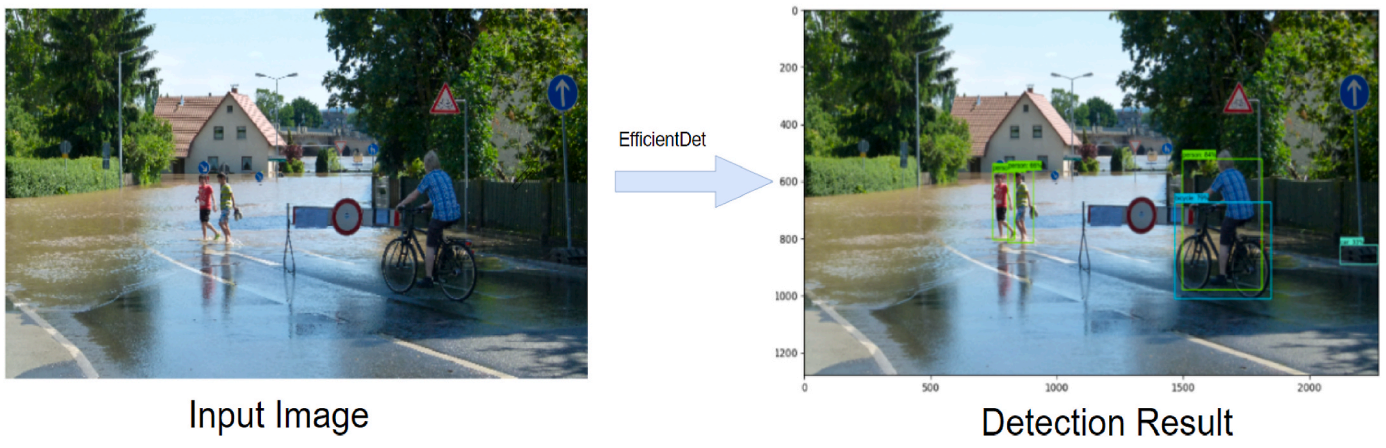

Fig. 10. EfficientDet detection results with bounding boxes.

Table 3
The prediction score of different object detection models using collected flood dataset. N/A= Not applicable.

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

Table 4
The prediction score of different object detection models using MSCOCO dataset.

Models/Object Categories	vehicle	Forest	Traffic Sign	Tree	Residential area	Person	water vessels	Critical Infrastructure (bridge, dam, road, storm water facilities, railroad)
SSD MobileNet	70%	N/A	36%	N/A	N/A	41%	30%	N/A
Fast R–CNN	99%	N/A	95.9%	N/A	N/A	98	45%	N/A
Mask R–CNN	98%	N/A	94%	N/A	N/A	99%	99%	N/A
YOLOv3	99.4%	N/A	99.96%	N/A	N/A	96.9%	94.9%	N/A
EfficientDet	81%	N/A	87%	N/A	N/A	86%	45%	N/A

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 and employing Feature Pyramid Networks (FPN; see Lin et al., 2017a,b) 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 can be found in different scales. To make the model robust to such scale changes, the model can 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

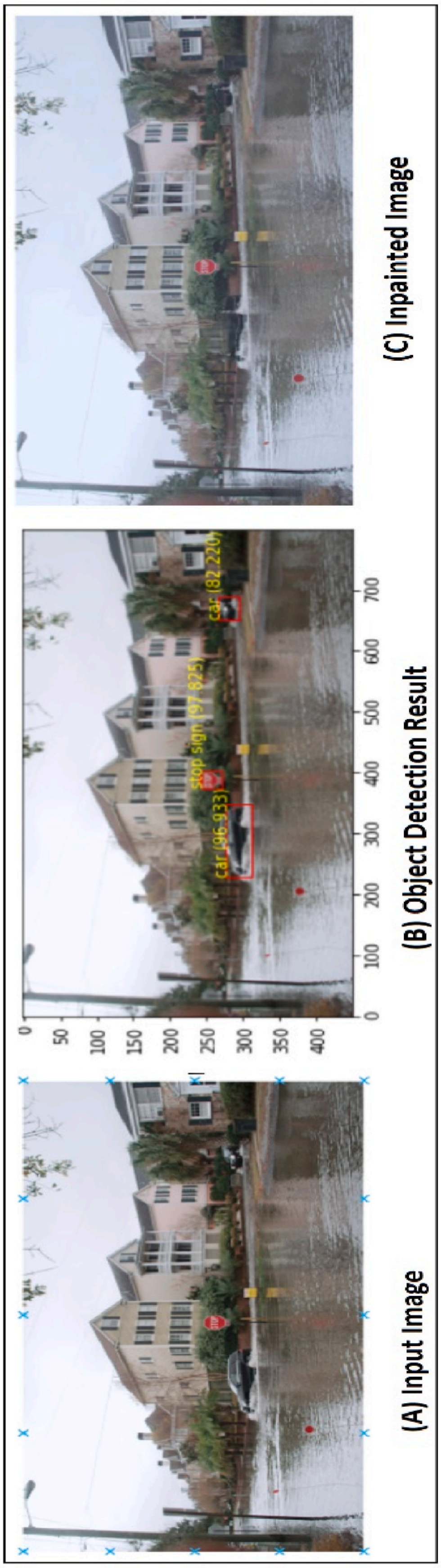

Fig. 11. Detection and inpainting pipeline.

Table 5
Water levels with associated aspect ratios and flood severity and risk estimation.

Water Level	Aspect Ratio	Flood Severity and Risk
Level 1	>1.8	Mild
Level 2	1.62–1.8	
Level 3	1.44–1.62	
Level 4	1.26–1.44	
Level 5	1.08–1.25	Moderate
Level 6	0.90–1.08	
Level 7	0.72–0.90	
Level 8	0.54–0.72	
Level 9	0.36–0.53	Severe
Level 10	0.18–0.36	
Level 11	<0.18	

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 here 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., 2017a,b) with an optimal network architecture. The third challenge is the extension of 2D object detection methods for object detection within videos (in case of the USGS river cam 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 LSTM (Long Short-Term Memory; 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 for real-time flood risk assessment. 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 with 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. In future work, we will concentrate on assessing flood risk and severity more quantitively and further explore the importance of video footage in real time road flooding assessment and monitoring.

Software and data availability

FloodImageClassifier.py: Various Scripts as they apply to image processing and flood label detection development.

Description: Image processing algorithms designed for flood image label detection and floodwater level estimation.

Developer: J. R. Pally and S. Samadi.

Contact: jpally@clemson.edu and samadi@clemson.edu.

Software Access: https://github.com/HHRClemson/FloodImageClassifier.

Year First Available: November 2021; Operating systems supported: Windows, Linux, MacOS.

Hardware required: Intel i5 or mid-performance PC with multicore processor and SSD main drive, 4 Gb memory recommended.

Cost: Free. Software and source code are released under the Massachusetts Institute of Technology License.

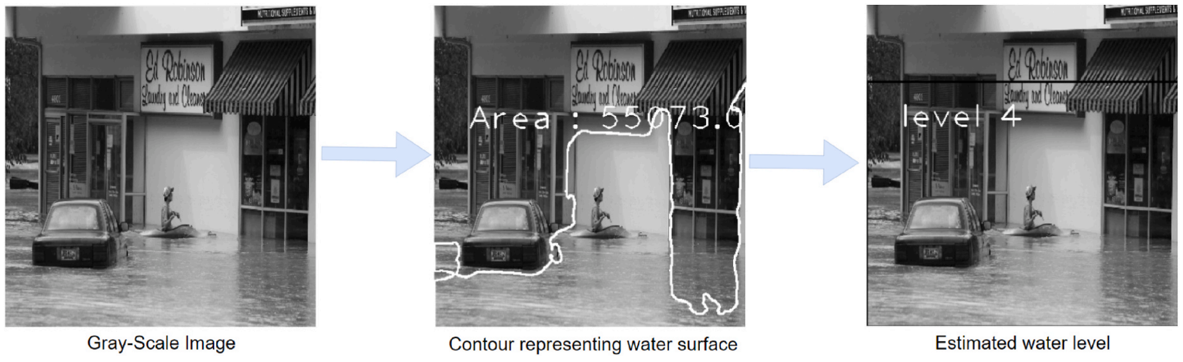

Fig. 12. Mild water depth estimation (level 4, see Table 5) using Canny Edge Detection and aspect ratio approaches along with calculated area which was 55073.6 pixels.

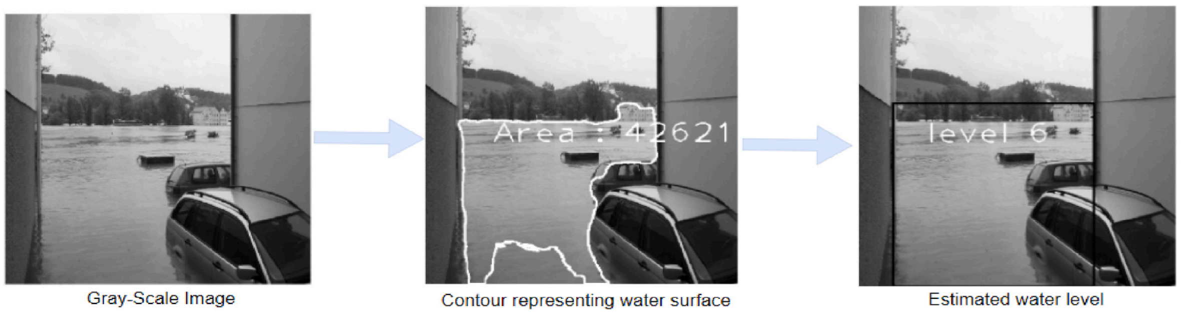

Fig. 13. Moderate water depth estimation using Canny Edge Detection and aspect ratio approaches along with calculated area which was 42621 pixels.

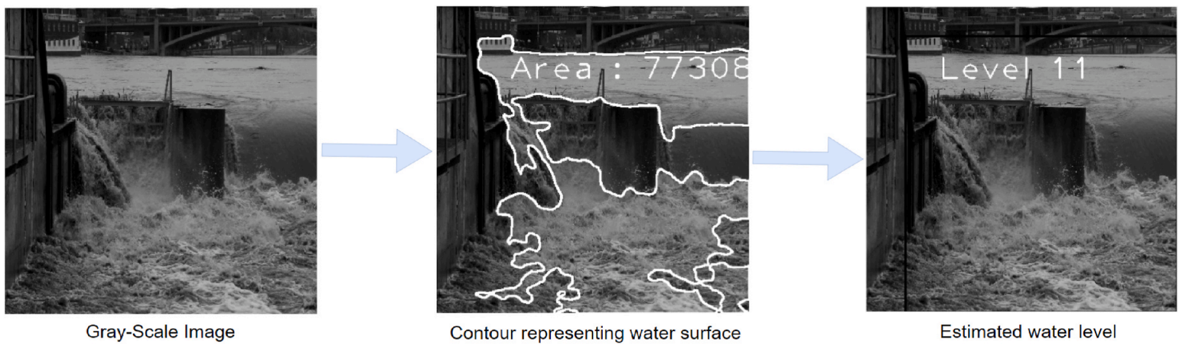

Fig. 14. Sever water depth estimation using Canny Edge Detection and aspect ratio approaches along with calculated area which was 77308 pixels.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

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.

References

Ali, Hashir, Mahrukh, Khursheed, Syeda Kulsoom, Fatima, Muhammad, Shuja Syed, Shaheena, Noor, 2019. Object recognition for dental instruments using SSD-MobileNet. In: International Conference on Information Science and Communication Technology. ICISCT), 2019.
Bantupalli, K., Xie, Y., 2018. December. American sign language recognition using deep learning and computer vision. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE, pp. 4896–4899.
Bay, H., Tuytelaars, T., Van Gool, L., 2006. Surf: Speeded up robust features. In: European Conference on Computer Vision. Springer, Berlin, Heidelberg, pp. 404–417.
Bouchakwa, M., Ayadi, Y., Amous, I., 2020. A review on visual content-based and users' tags-based image annotation: methods and techniques. Multimed. Tool. Appl. 79 (29), 21679–21741.
Brownlee, J., 2019. How to Perform Object Detection with YOLOv3 in Keras. Available at: https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/. accessed on November 11, 20.
Brunetti, A., Buongiorno, D., Trotta, G.F., Bevilacqua, V., 2018. Computer vision and deep learning techniques for pedestrian detection and tracking: a survey. Neurocomputing 300, 17–33.
Canny, J., 1986. A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. (6), 679–698.

Chaudhary, P., D'Aronco, S., Moy de Vitry, M., Leitão, J.P., Wegner, J.D., 2019. Flood-water level estimation from social media images. ISPRS Ann. Photogram. Rem. Sens. Spatial Inform. Sci. 4 (2/W5), 5–12.

Chemelil, P.K., 2021. Single Shot Multi Box Detector Approach to Autonomous Vision-Based Pick and Place Robotic Arm in the Presence of Uncertainties. Doctoral dissertation, JKUAT-COETEC.

Chollet, F., et al., 2015. Keras. GitHub. Retrieved from. https://github.com/fchollet/keras.

Clark, S., 2019. The Design and Prototyping of Innovative Sustainable Material Solutions for Automotive Interiors. Royal College of Art (United Kingdom).

Criminisi, A., Perez, P., Toyama, K., 2003. June. Object removal by exemplar-based inpainting. In: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings, Vol. 2. IEEE, p. II.

Croitoru, I., Bogolin, S.V., Leordeanu, M., 2017. Unsupervised learning from video to detect foreground objects in single images. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 4335–4343.

Donratanapat, N., Samadi, S., Vidal, M.J., Sadeghi Tabas, S., 2020. A national-scale big data prototype for real-time flood emergency response and management. Environ. Model. Softw. https://doi.org/10.1016/j.envsoft.2020.104828.

Erhan, D., Szegedy, C., Toshev, A., Anguelov, D., 2014. Scalable object detection using deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2147–2154.

Francalanci, C., Guglielmino, P., Montalcini, M., Scalia, G., Pernici, B., 2017. May. IMEXT: a method and system to extract geolocated images from Tweets—analysis of a case study. In: 2017 11th International Conference on Research Challenges in Information Science (RCIS), pp. 382–390. IEEE. [17] [15] Freund, Y. and Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1), pp.119–139.

Fu, C.Y., Liu, W., Ranga, A., Tyagi, A., Berg, A.C., 2017. Dssd: Deconvolutional Single Shot Detector arXiv preprint arXiv:1701.06659.

Girshick, R., 2015. Fast r-cnn. In: Proceedings of the IEEE International Conference on Computer Vision. https://doi.org/10.1109/ICCV.2015.169.

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587.

He, K., Gkioxari, G., Dollár, P., Girshick, R., 2017. Mask r-cnn. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2961–2969.

He, Kaiming, Gkioxari, Georgia, Dollar, Piotr, Girshick, Ross, 2020. Mask R-CNN. IEEE Trans. Pattern Anal. Mach. Intell.

Hochreiter, S., Schmidhuber, J., 1997. LSTM can solve hard long time lag problems. Adv. Neural Inf. Process. Syst. 473–479.

Jaderberg, M., Simonyan, K., Zisserman, A., 2015. Spatial transformer networks. Adv. Neural Inf. Process. Syst. 28, 2017–2025.

Kang, K., Li, H., Xiao, T., Ouyang, W., Yan, J., Liu, X., Wang, X., 2017. Object detection in videos with tubelet proposal networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 727–735.

Kharazi, B.A., Behzadan, A.H., 2021. Flood depth mapping in street photos with image processing and deep neural networks. Comput. Environ. Urban Syst. 88, 101628.

Kharazi, B.A., Behzadan, A.H., 2021. Flood depth mapping in street photos with image processing and deep neural networks. Comput. Environ. Urban Syst. 88, 101628.

Kong, T., Sun, F., Yao, A., Liu, H., Lu, M., Chen, Y., 2017. Ron: reverse connection with objectness prior networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5936–5944.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444.

Li, J., Huang, Y., Xu, Z., Wang, J., Chen, M., 2017a. July. Path planning of UAV based on hierarchical genetic algorithm with optimized search region. In: 2017 13th IEEE International Conference on Control & Automation (ICCA), pp. 1033–1038.

Li, Q., Jin, S., Yan, J., 2017b. Mimicking very efficient network for object detection. In: Proceedings of the Ieee Conference on Computer Vision and Pattern Recognition, pp. 6356–6364.

Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C. L., 2014. September. Microsoft coco: common objects in context. In: European Conference on Computer Vision. Springer, Cham, pp. 740–755.

Lin, T., Goyal, P., Girshick, R., He, K., Dollár, P., 2017a. Focal loss for dense object detection. In: IEEE International Conference on Computer Vision, pp. 2999–3007.

Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017b. Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2117–2125.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C., 2016a. October. Ssd: single shot multibox detector. In: European Conference on Computer Vision. Springer, Cham, pp. 21–37.

Liu, Wei, Anguelov, Dragomir, Erhan, Dumitru, Szegedy, Christian, Scott, Reed, Fu, Cheng-Yang, 2016b. SSD: Single Shot MultiBox Detector" arXiv:1512.02325.

Liu, P., Hongbo, Y.A.N.G., Hu, Y., Fu, J., 2018. November. Research on target recognition of underwater robot. In: 2018 IEEE International Conference on Advanced Manufacturing (ICAM). IEEE, pp. 463–466.

Lowe, D.G., 1999. Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision.

Najibi, M., Rastegari, M., Davis, L.S., 2016. G-cnn: an iterative grid-based object detector. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2369–2377.

Nie, S., Zheng, M., Ji, Q., 2018. The deep regression bayesian network and its applications: probabilistic deep learning for computer vision. IEEE Signal Process. Mag. 35 (1), 101–111.

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

Ning, H., Li, Z., Hodgson, M.E., Wang, C.S., 2020. Prototyping a social media flooding photo screening system based on deep learning. ISPRS Int. J. Geo-Inf. 9 (2), 104.

Pan, J., Yin, Y., Xiong, J., Luo, W., Gui, G., Sari, H., 2018. Deep learning-based unmanned surveillance systems for observing water levels. Ieee Access 6, 73561–73571.

Papandreou, G., Zhu, T., Kanazawa, N., Toshev, A., Tompson, J., Bregler, C., Murphy, K., 2017. Towards accurate multi-person pose estimation in the wild. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4903–4911.

Park, S., Baek, F., Sohn, J., Kim, H., 2021. Computer vision–based estimation of flood depth in flooded-vehicle images. J. Comput. Civ. Eng. 35 (2), 04020072.

Phillips, R.C., Samadi, S.Z., Meadows, M.E., 2018. How extreme was the October 2015 flood in the Carolinas? An assessment of flood frequency analysis and distribution tails. J. Hydrol. 562, 648–663.

Powers, D.M., 2020. Evaluation: from Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation arXiv preprint arXiv:2010.16061.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788.

Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. Adv. Neural Inf. Process. Syst. 28, 91–99.

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., Savarese, S., 2019. Generalized intersection over union: a metric and a loss for bounding box regression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 658–666.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., 2015. Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. 115 (3), 211–252.

Schneider, S., Taylor, G.W., Kremer, S., 2018. May. Deep learning object detection methods for ecological camera trap data. In: 2018 15th Conference on Computer and Robot Vision. CRV), pp. 321–328.

Shen, Z., Liu, Z., Li, J., Jiang, Y.G., Chen, Y., Xue, X., 2017. In: Dsod: learning Deeply Supervised Object Detectors from Scratch. ICCV.

Simonyan, K., Zisserman, A., 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition arXiv preprint arXiv:1409.1556.

Sivic, J., Zisserman, A., 2003. October. Video Google: a text retrieval approach to object matching in videos. In: Computer Vision, IEEE International Conference on, vol. 3. IEEE Computer Society, p. 1470.

Tan, M., Pang, R., Le, Q.V., 2020a. Efficientdet: scalable and efficient object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10781–10790.

Tan, Mingxing, Pang, Ruoming, Le Quoc, V., 2020b. EfficientDet: scalable and efficient object detection. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

Vincent, O.R., Folorunso, O., 2009. June. A descriptive algorithm for sobel image edge detection. In: Proceedings of Informing Science & IT Education Conference (InSITE), vol. 40, pp. 97–107.

Wang, C., Ren, W., Huang, K., Tan, T., 2014. September. Weakly supervised object localization with latent category learning. In: European Conference on Computer Vision. Springer, Cham, pp. 431–445.

Xu, Beibei, Wang, Wensheng, Greg, Falzon, Paul, Kwan, Guo, Leifeng, Chen, Guipeng, Amy, Tait, Schneider, Derek, 2020. Automated cattle counting using Mask R-CNN in quadcopter vision system. Comput. Electron. Agric. 171, 105300.

Xu-kai, Z., Qiong-qiong, L., Baig, M.H.A., 2012. June. Automated detection of coastline using Landsat TM based on water index and edge detection methods. In: 2012 Second International Workshop on Earth Observation and Remote Sensing Applications. IEEE, pp. 153–156.

Yang, H.C., Wang, C.Y., Yang, J.X., 2014. Applying image recording and identification for measuring water stages to prevent flood hazards. Nat. Hazards 74 (2), 737–754.

Yang, Hongbo, Liu, Ping, Hu, YuZhen, Fu, JingNan, 2020. Research on underwater object recognition based on YOLOv3. Microsyst. Technol. 27, 1837–1844.

Yiatrou, P., Polycarpou, I., Read, J.C., Zeniou, M., 2016. September. The synthesis of a unified pedagogy for the design and evaluation of e-learning software for high-school computing. In: 2016 Federated Conference on Computer Science and Information Systems (FedCSIS). IEEE, pp. 927–931.

Yoo, D., Park, S., Lee, J.Y., Paek, A.S., So Kweon, I., 2015. Attentionnet: aggregating weak directions for accurate object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2659–2667.

Zhao, Z.Q., Xie, B.J., Cheung, Y.M., Wu, X., 2014. November. Plant leaf identification via a growing convolution neural network with progressive sample learning. In: Asian Conference on Computer Vision, pp. 348–361.

Zhao, Z.Q., Zheng, P., Xu, S.T., Wu, X., 2019. Object detection with deep learning: a review. IEEE Trans. Neural Network. Learn. Syst. 30 (11), 3212–3232.

Zhu, Z., Brilakis, I., Parra-Montesinos, G., 2009. Real-time concrete damage visual assessment for first responders. Construction Research Congress 2009: Building a Sustainable Future 1204–1213.