UAV-assisted Aerial Survey of Railways using Deep Learning

Dimitrios Kafetzis*, Ioannis Fourfouris*, Savvas Argyropoulos† and Iordanis Koutsopoulos*
*Athens University of Economics and Business, †StreamOwl

***Abstract*—Unmanned Air Vehicles (UAVs) are currently in use for diverse applications such as critical infrastructure monitoring. Monitoring is based on video capture by a video camera and subsequent use of Deep Learning (DL) techniques to perform image recognition. In this paper we do a proof-of-concept validation of DL and UAV-assisted monitoring for vegetation and object detection on railways. The large diversity of vegetation, terrain and railway settings increases the challenges for object detection and classification. Moreover, the creation of an appropriate dataset for the training of a classifier is a nontrivial task per se. We show the related challenges in this setting, and we create from scratch a dedicated dataset with manual annotation for vegetation management on railways, based on publicly available video clips and our own video recordings at a railway. To the best of our knowledge this is the first dedicated dataset for this application. Next, we develop a DL pipeline on this dataset and evaluate its performance for different classes of vegetation and obstacles on the railways. Our approach leads to satisfactory detection accuracy, especially given the diversity of obstacles and the fact that most objects to be detected appear at the background of the frame image. Also, we test our classifier models in the NVIDIA Jetson Nano platform, which is the on-board computing system of a UAV that we used for on-site testing. The classifier may operate on the Jetson Nano board presenting a good and viable performance.**

I. Introduction

Managing lineside vegetation trees, bushes, weeds and obstacles (e.g. fallen trees, fallen branches, garbage, etc) on railways is an important task accidents for passing trains, railway staff and passengers. It is also important for train drivers so that they have a clear view of the route ahead, including signals and crossings, and for track workers so that they are safe when trains and maintenance vehicles pass. Hence the removal of vegetation within 3.5 metres of the line is an essential preventive safety measure that is put in action based on related regulations.

Network Rail in the UK, a major railway operator, claims that incidents caused by vegetation cost the railway company more than £100M a year [1], [2]. For this reason, railway operators do frequent inspections of railway tracks in order to ensure that the operation of trains is safe and efficient. On the other hand, unnecessary de-vegetation damages green railway corridors [3]. For example, Network Rail has already issued tree preservation orders on a group of trees. Therefore, there is need for a systematic smart and efficient procedure for inspection of railways.

Currently railway operators follow traditional railway track inspection methods, based on human inspectors and ground vehicles. Optical inspection, which involves investigation of various types of rail components and vegetation, is the main practice followed. This human-operated process by default cannot be performed very often and it is time-inefficient, costly and complex. Also, workers are faced with slopes, barriers, unstable ground, hidden hazards and poor lighting to carry out visual inspections. The landscape and weather conditions in some areas pose additional challenges to human operators.

The goal of this paper is to study and demonstrate the application of DL techniques in computationally difficult problems such as that of vegetation and obstacle detection on railways, performed at the edge of a network, namely at the on-board computing system of a UAV. The main reasons that make the problem challenging are: (*i*) the vastly heterogeneous terrains, (*ii*) the multiple classes of vegetation and obstacles that need to be distinguished, (*iii*) the diverse depths (in the image) of the position of objects to be detected (i.e. the objects that we want to detect more often than not appear in the background) and (*iv*) the issue of accurately tracking the railway, so that the UAV follows the rails track.

A. Our contribution

We summarize our contributions as follows:

- We create from scratch a dedicated dataset, which involves manual image annotation, for vegetation management on railways. The dataset is used for the training of Convolutional Neural Networks (CNNs) in order to analyse video frames captured by a drone's camera and to detect obstacles on the rails, as well as vegetation and trees growing close to the rails. The dataset is based on publicly available videos and videos recorded by us.
- We select two different versions of the Faster RCNN model, the *Faster RCNN Inception v2* model that is the basic version of this algorithm, and the *Faster RCNN Inception ResNet v2 Atrous* that is an optimized

This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant (Project Number: HFRI-FM17-352). It was also supported from European Union's H2020 program ESMERA EREVOS project through University of Patras. I. Koutsopoulos also acknowledges support from a GSRT Research Reinforcement grant for the project net-commons, and from the AUEB grant "Original Scientific publications". The authors wish to thank Mr. Zissis Biloroglou for his help with dataset processing.

version for small-scale object detection. We evaluate and compare them in terms of detection accuracy.
- We train a DL classifier, based on the above mentioned models, that is able to achieve very good up to excellent performance in the following tasks: (*i*) recognition and tracking of railway lines so that they are continuously inspected by a UAV resulting at 99% average precision (*ii*) detection, identification and classification of possible obstacles on rails at about 64% to 79% average precision, and (*iii*) detection and classification of vegetation on, between and out of rails with about 63% to 93% average precision.
- We test the classifier on a real railway. This gives us a better understanding for its performance under real conditions on site testing.

The rest of the paper is organized as follows. Section II reviews previous research. The creation of the dataset used for the training and evaluation of the classifier, the DL classifier's model and its performance evaluation results are presented and discussed in Section III. Further extensions which are related with the evaluation of a special version of the classifier's model and the evaluation of the classifier's performance at the on-board processor are presented in Section IV. Finally, we draw conclusions in Section V and describe our next steps.

II. Related Work

Computer vision has been a critical component in rail track inspection system development. In [4], Karakose et al. have proposed a computer vision-based condition monitoring method. In this study, a camera placed on top of the train, collects images and afterwards the system applies edge extraction, morphological processing and feature extraction on the obtained images to determine rail failures. Computer vision has been used in drone image process in [5]. In [6], a system with four Charge-coupled Device (CCD) cameras and two red laser sector lights for inspection of track gauge based on computer vision is used. In [7], Camargo et al. have conducted a survey, aiming to investigate the feasibility of using machine vision technology to recognize turnout components, cut spikes and rail anchors and show performance measurement for the algorithms which find defects in other track components.

Several different techniques have been used in order to make rail track inspection more efficient. In [8], Gabor filters are applied to images, and they transform them into directionally filtered versions to distinguish various track areas such as track turnouts. A system for detecting missing rail clips and finding blue rail clips which have been recently replaced in place of damaged rail clips, using automated video analysis, is proposed in [9]. In [10], Teng et al. have presented a learning based algorithm which detects railway regions based on Support Vector Machine (SVM). A combination of multiple detectors within a multi-task learning framework,

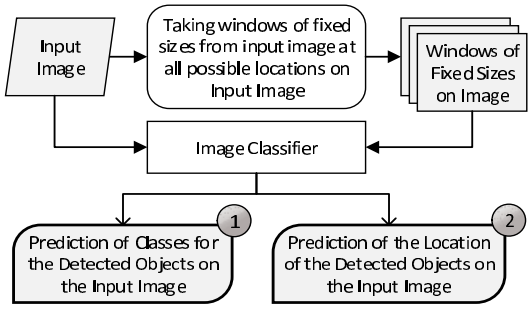

FIG. 1: Basic processing pipeline for object detection in an image.

showed the improvement of defects' detection on railway ties and fasteners in [11].

Current developments in machine learning for computer vision and the continuous improvement of the computational power of low-weight systems that can be integrated into a UAV with easy portability look very promising for real-time systems implementation. Transfer learning and data augmentation techniques have been used for training deep learning models, focusing on identifying track defects such as sunkinks, loose ballast and railway assets like switches and signals [12]. In [13], a high speed 3-D laser range finder helps towards the development of a real-time railway fastener detection system from depth images. In [14], James et al. have proposed a multiphase deep learning based technique that finds the Regions of Interest (ROIs) with image segmentation and feeds a classifier with the cropped image to identify and classify the rail surface defects. An image acquisition device equipped with LED auxiliary light source and shading box for real-time visual detection of a rail surface, with emphasis on target location and rail surface defects contour extraction, is presented in [15]. Our work complements the current literature by presenting created dedicated dataset for railway image processing, and a proof-of-concept validation for computer vision and image recognition techniques vegetation and obstacle identification in railways.

III. A Deep Learning Classifier for Vegetation Management On Railways

Object detection can be modelled as a classification problem. The basic processing pipeline for object detection is depicted in Fig. 1. The main challenges of the classification procedure are related to:

a. the different size of the window that always contains the object, and
b. the aspect ratio of objects. Aspect ratio concerns the relationship between width and height of an object. An object can be presented in various aspect ratios.

Specifically in this application scenario, the image recognition problem is focused on detection of specific types of vegetation and obstacles on railways, through aerial survey in an autonomous manner using a UAV. It is a nontrivial problem because of the following reasons:

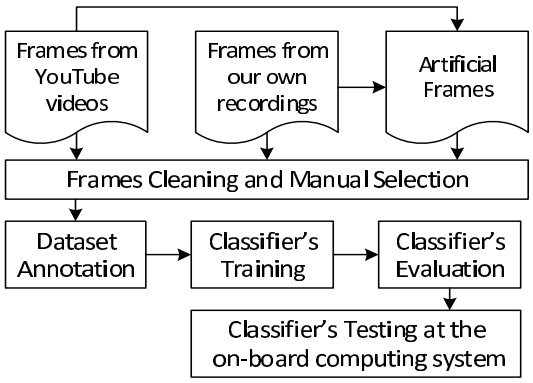

FIG. 2: Pipeline for the proposed approach.

a. Various types of vegetation and obstacles on railways based on the point of view (POV) of the UAV-mounted camera are in the background of the captured video frames. Thus, objects may be not well focused in an image.
b. For the tracking of rails we have to follow an on-line approach, through processing the video frame by frame. This is a challenging task because of the limited resources of the on-board processor.
c. The handling of blurred/moved images due to occasional minor deviating moves of the drone is another challenge to overcome.
d. The creation of a dedicated dataset for this application is a difficult task per se, and there are no such datasets publicly available.

The approach of our methodology is shown in Fig. 2.

A. Datasets and annotation

The methodology for detection and classification of vegetation and objects on railways is based on supervised machine learning techniques for computer vision. This methodology requires a dataset with annotated images of objects, similar to those that our classifier has to detect. This dataset will be used for the training of the classifier's model. To the best of our knowledge, a dedicated dataset for training of Deep Learning (DL) models for obstacle detection and vegetation management on railways does not exist. Therefore we started off to create dedicated datasets from scratch by collecting images from public videos from two countries and by recording our own videos. For the creation of the dataset, we took into account that we needed diversity on exogenous conditions e.g.: (*i*) the landscapes themselves, (*ii*) vegetation and objects on railways to be detected and recognized, and (*iii*) weather conditions (cloudy or sunny landscapes).

We created two datasets, dataset DS1 that is based on YouTube publicly available videos [16], [17], and dataset DS2 that is based on our own recordings on an abandoned railway in Greece. Further, in our evaluation sessions we considered the use of a third dataset, DS3, that emerged through merging DS1 and DS2. Table I shows some details

Classes	Number of data-points in DS1	Number of data-points in DS2
"weed on rails"	377	894
"object on rails"	229	212
"tree on rails"	18	131
"rails"	6,415	1,107
"fence"	1,160	75
"bush"	3,500	2,675
"terrain"	2,400	33
"tree"	4,199	824
"weed between rails"	233	381
"weed out of rails"	633	1,183
"branches above rails"	13	1,011
"branches close to rails"	10	581

TABLE I: Number of examples per label and per dataset. DS1 is the publicly available dataset collected and processed from YouTube videos, featuring UK and Slovenia landscapes. DS2 refers to our own created dataset captured in Tripoli (Greece).

Dataset	Origin	Comments
DS1	Public	Based on YouTube videos
DS2	Our own recordings	Based on our own recordings in Tripoli (Greece)
DS3	Public and Private	Combination of DS1 and DS2

TABLE II: An explanation of how the datasets that we use are created. Dataset DS1 is the publicly available dataset collected and processed from YouTube videos, featuring UK and Slovenia landscapes and dataset DS2 is created totally from scratch based on our video recordings in Tripoli (Greece).

about our datasets DS1, DS2 and DS3. The datasets include annotated images taken from the video clips as isolated frames. The annotations on the images refer to the following 12 classes of objects and obstacles: (*a*) "bush", (*b*) "tree", (*c*) "fence", (*d*) "terrain", (*e*) "weed between rails", (*f*) "weed on rails", (*g*) "weed out of rails", (*h*) "object on rails", (*i*) "tree on rails", (*j*) "branches above rails", (*k*) "branches close to rails" and (*l*) "rails". Details about the number of datapoints per class for the datasets DS1 and DS2 are included on Table II.

Dataset DS1 includes 2,484 annotated images taken from publicly available recorded videos in Slovenia and UK with resolution 1280x720 pixels. Dataset DS2 includes 1,425 annotated images from our own recordings in Greece with resolution 1920x1080 pixels. These videos allowed for high diversity of terrains with respect to the color tones, brightness, objects and obstacles. This is prerequisite for successful training of an object detection CNN model that acts as the railway and vegetation classifier. The separate frames from the videos, which were inserted into the datasets after the annotation procedure, fulfil the following criteria: (*i*) high diversity of typical examples as to the objects of each class, (*ii*) high clarity images of typical examples (*iii*) diversity in terms of weather conditions and levels of brightness.

We annotated the video frames using frame isolation,

FIG. 3: Example image with its annotations from dataset DS1.

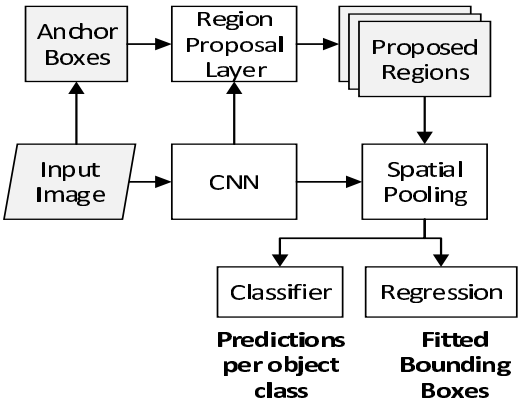

FIG. 4: Faster RCNN processing pipeline for object detection in an image.

dataset pre-processing and subsequent cleaning. We selected frames with typical case examples that we want to use for the training of the DL object detection algorithm in order to identify similar cases. Repetitive as well as problematic images (e.g. blurry, wrongly focused, distorted) were removed because the object classes on these frames were unclear and difficult to recognize.

The dataset consists of two subsets, the training dataset and the testing dataset that were used for the evaluation of the classifier after training. According to best practices for object detection models [18], the training dataset size is 80% of the size of total dataset and that of the test dataset is the rest 20%. Specifically, in dataset DS1, 1,989 images are in the training dataset and 495 annotated images are in the test dataset. Dataset DS2 consists of 1,140 annotated images for the training dataset and 285 annotated images for the test dataset.

The implementation of the classifier model is based on **Tensorflow** [19], Google's open source library for numerical computation and large scale machine and deep learning. Therefore, the training and test datasets were transformed to `.tfrecord` files, which can be processed by the Tensorflow's algorithms. The datasets are structured under the well-known and well-used dataset format, **MS COCO** [20].

For the creation of dataset we followed two steps: (*i*) image extraction from videos, and manual selection of suitable images and (*ii*) annotation. The latter step is that of assigning labels to objects in the images of our custom dataset that we want the DL classifier to recognize. The labels of our dataset are the 12 classes on Table II. For the isolation and annotation of the video frames, two software tools were used. The first one is **ffmpeg** [21], for the extraction of all frames of videos as JPEG images, and the second one is **labelme** [22] that was used for annotation. Annotation was based on polygons that turn out to capture well complex and irregular shapes of objects like trees, bushes and weeds.

The dataset was created under the condition that the collection of videos that will be used as the main source of the dataset will have similar or the same POV with the drone's camera. Some classes such as "weed on rails", "object on rails", "tree on rails", "branches above rails" and "branches close to rails" were not populated enough in natural video frames. Therefore, in order to ensure sufficient representation of each class in the dataset, we created custom artificial images using image processing software tools like Adobe Photoshop [23]. With this method, our dataset is augmented in specific classes where the natural example classes were rare. An example of an image with its annotations from our dataset appears in Fig. 3.

B. Object Detection Model Selection

A CNN consists of several layers such as the input layer, at least one hidden layer, and an output layer. They are used in object detection for recognizing patterns such as edges (vertical/horizontal), shapes, colors, and textures. Hidden layers are convolutional layers, which work like a filter that first receives input, transforms it using a specific pattern/feature, and sends it to the next layer. For example, in the first convolutional layer, the filter may identify shape/color in a region (e.g. green). The next layer may be able to conclude what the object what is (e.g. a trunk with branches), and the last convolutional layer may classify the object as a tree. More sophisticated patterns can be detected as more layers are traversed.

Faster RCNN [24] is a single, unified network for object detection and object classification and it is composed by two modules. The first one is a deep full convolutional network that proposes regions - that is, the Region Proposal Network (RPN) - and the second module is the Fast RCNN [25],[26] detector that uses the proposed regions. The RPN, which takes an image as input and outputs a set of rectangular object proposals, each with an objectness score, is the module that tells the Fast RCNN module where to look. In order to train RPNs, we assign a binary class label (corresponding to existence of an object or not) to each anchor. Anchors are reference boxes for each proposed region by the RPN. The RPN can be trained end-to-end by back-propagation and stochastic gradient descent (SGD). The processing pipeline of the Faster RCNN algorithm is shown on Fig. 4.

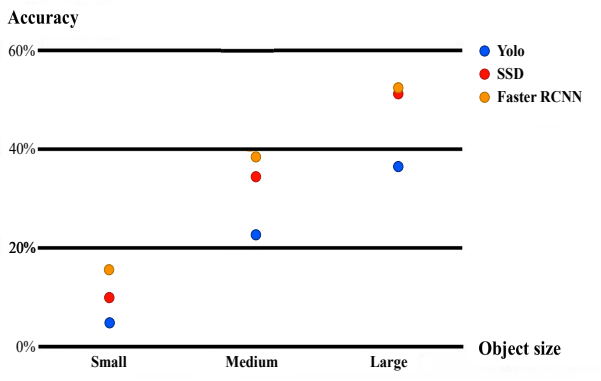

FIG. 5: Accuracy vs object size for different object detection models.

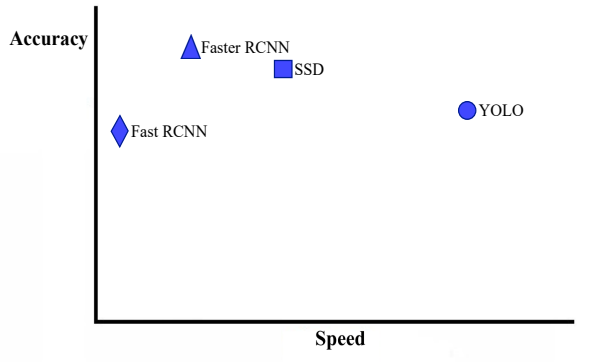

FIG. 6: Accuracy vs Training speed for different object detection models.

Other object detection models are the Single Shot Detector one (SSD) and the You Only Look Once one (YOLO). In SSD [27], the tasks of object localization and classification are done in a single forward pass of the network. In YOLO [28], each image is divided into a grid and each grid predicts N bounding boxes and includes a confidence value. The confidence reflects the accuracy of the bounding box and whether it actually contains an object (regardless of class). YOLO predicts the classification score for each box for each class in training.

Our choice for the DL model as the obstacle and vegetation classifier is the *Faster RCNN* model. The main reasons are:

a. Faster RCNN enables object detection at near real-time frame rates.
b. It provides high object detection accuracy. (Figs 5, 6)
c. It is trained faster. (Fig. 6)

C. Training

Faster RCNN is a combination of Fast RCNN that has the role of detector, and RPN that has the role of region proposer. The training procedure has 4 steps:

a. RPN training, which proposes the input image anchors, where the detected objects have high probability to be there. In this step, convolutional layers are initialized with the Inception pre-trained model.
b. A separate detection network is trained by Fast RCNN using the proposals generated by the first step, and the RPN that is initialized by the Inception pre-trained model.
c. The convolutional layer optimizes RPN that is initialized by detector network in the second step.
d. The convolutional layer optimizes Fast RCNN. The training loss, which is the prime indicator of accuracy of training is tracked and reduced in successive iterations of stochastic gradient descent (SGD) [29].

The main training properties and hyper-parameters of the Faster RCNN algorithm are the default, as follows:

a. the weighting values are normally distributed with mean 0 and variance 0.01, thus they are initialized with $N(0, 0.01^2)$;
b. the learning rate is 0.0002 for 500,000 steps;
c. the learning update scheme is based on a momentum update 0.9. Momentum changes the path that SGD takes to the optimum point during training and it helps overcome local optimum if the algorithm gets stuck; and
d. the training steps are 500,000. This is the value for which we had the best results.

In our training sessions we used the *faster_rcnn_inception_v2_coco* model, downloaded from the *Tensorflow detection model zoo* [30]. The training procedure is running on a PC with a CPU Intel Core i7-5820K (15Mb Cache, up to 3.6GHz), RAM 16GB and a GPU NVIDIA Quadro M5000. The average training time was approximately 42 hours for 500,000 training steps. The same PC was used for the performance evaluation which is presented in the sequel.

D. Performance evaluation metrics

Precision for a given class c in classification, is the ratio of the number of true positives (TP_c) and the number of predicted positives, where the definition of true positives, false positives, true negatives and false negatives is given in the confusion matrix in Table III. It is,

$$Precision_c = \frac{TP_c}{TP_c + FP_c} \tag{1}$$

The recall of a given class c in classification, is defined as the ratio of TP_c and total of ground truth positives and it is given by

$$Recall_c = \frac{TP_c}{TP_c + FN_c} \tag{2}$$

The main performance evaluation metrics in object detection are the Mean Average Precision (mAP) across all classes (labels) and the Average Precision (AP_c) per class. To understand mAP, we have to define first the term *precision-recall* curve, a commonly used to measure the performance of

		Actual	
		Positive	Negative
Predicted	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

TABLE III: Confusion matrix for investigation of the performance of a classification model where the actual test values are known.

Train Dataset \ Test Dataset	DS1	DS2	DS3
DS1	S(1,1)	S(1,2)	S(1,3)
DS2	S(2,1)	S(2,2)	S(2,3)
DS3	S(3,1)	S(3,2)	S(3,3)

TABLE IV: Evaluation scenarios. $S(i, j)$ denotes the case where the training dataset is from DS_i and the test dataset is from DS_j, for $i, j = 1, 2, 3$.

a classification model. A precision-recall curve has on y-axis the precision and on x-axis the recall. Intuitively, the area under the curve, namely the integral of the function denoted by the curve shows the infinite sum of achieved accuracy values for a continuous set of recall values. The AP_c is defined as the integral under the precision-recall curve (PR curve).

There exists a trade-off between precision and recall. On the one hand, we want our precision to be as high as possible, thus we would need to decrease FP; however by doing so, recall will be decreased as well. Similarly, by decreasing FN, recall is increased and precision is decreased as well. In object detection cases like the one we consider here, we would like our precision to be high (namely, our predicted positives to be close to TP).

To get the AP values for the defined classes, we plotted the Precision vs Recall curves for each class. Specifically we take a discrete set of Q recall values $(r_1, ..., r_Q)$ and we record the precision at these values (i.e. Precision(r_i)) for $i = 1, \ldots, Q$. The number Q is equal to the number of the images where the given class is detected. Some indicative examples of PR curves from our evaluations are presented in *Appendix I* (Fig. 10). The AP for a given class c is computed as

$$AP_c = \frac{\sum_{i=1}^{Q} Precision_c(r_i)}{Q}, \tag{3}$$

The mAP over all classes is computed as

$$mAP = \frac{\sum_{c=1}^{C} AP_c}{C}, \tag{4}$$

where C is the number of classes.

To calculate AP for object detection, we also need to define the term IoU (Intersection of Union). IoU is an estimate of the amount of overlap between 2 bounding boxes or segmentation masks. The IoU is a ratio, where the numerator shows the area of intersection of the predicted and the ground-truth bounding boxes, and the denominator shows the area of union of these boxes. IoU is used to determine if a predicted bounding box (BB) is a TP, a FP or a FN. The predicted BB is not evaluated if it is a TN because we assume that every image includes an object. The intuitive meaning of IoU is that IoUs with values $0 <$ IoU ≤ 0.5 are mostly for background objects and IoUs with $0.5 <$ IoU ≤ 1 are for foreground objects. Therefore, a challenge during the evaluation process was to select the appropriate IoU value, depending on the location of objects in an image in which we need to detect an object. In our case, we needed to evaluate our classifier for IoU values 0.1-0.4 because in railway monitoring, objects to be detected (e.g. tree, bushes) tend to be in the background. Specifically, we select $IoU = 0.1$ for the AP_c results.

E. Performance Evaluation Results

1) Evaluation Approach: First, we evaluate the generalizability of trained models. This allows us to conclude if we can have a trained model on a diverse dataset that can adapt to different terrains and exogenous conditions or we would have to create custom dataset depending on the scenario each time. To examine the generalizability and overall performance of our models, we evaluated 9 scenarios which are shown in Table IV.

Our strategy is to train and test in all 9 possible pairs of DS1, DS2, and DS3 and to measure the performance based on the average precision (AP) per class, mean average precision (mAP) and precision-recall curves for IoU values $0.1, 0.2, 0.3$ and 0.4. Denote by $S(i, j)$ the scenario where we use the training dataset from dataset DS_i and the test dataset from dataset DS_j for $i, j = 1, 2, 3$.

The training total loss per scenario, which is the prime indicator of accuracy of training is equal to 0.08.

2) Mean Average Precision Results: The mAP results (Table V) for the whole range of IoU shows that the classifier has a satisfactory performance when the training dataset and the test dataset are from the same dataset. The best results are at scenarios S(1,1), S(2,2) and S(3,3). On the other hand, we do not have good results when $i \neq j$, because of the high diversity between the landscapes and types of trees, bushes, terrains and weeds which are annotated at the images of datasets.

Also, at the scenarios where the training dataset or the test dataset come from the DS3, the performance of the classifier ranges from quite good to satisfactory. This is expected because DS3 includes datapoints from both DS1 and DS2. Based on the results (Table V) we can conclude that a model trained in a certain dataset cannot be used to perform testing in another dataset.

3) Indicative Average Precision per class Results: We present and discuss the AP results of classes "object on rails", "terrain" and "rails" for $IoU = 0.1$.

Object On Rails: The best results for class "object on rails" are for the scenarios where the training and test datasets are from the same dataset i.e. scenarios S(1,1), S(2,2) and

mAP @IoU=0.1			
Train Dataset \ Test Dataset	DS1	DS2	DS3
DS1	55%	10%	32%
DS2	12%	66%	37%
DS3	53%	57%	54%

mAP @IoU=0.2			
Train Dataset \ Test Dataset	DS1	DS2	DS3
DS1	51%	8%	29%
DS2	9%	62%	33%
DS3	48%	54%	49%

mAP @IoU=0.3			
Train Dataset \ Test Dataset	DS1	DS2	DS3
DS1	46%	6%	24%
DS2	6%	60%	30%
DS3	45%	51%	45%

mAP @IoU=0.4			
Train Dataset \ Test Dataset	DS1	DS2	DS3
DS1	40%	5%	21%
DS2	4%	55%	27%
DS3	41%	48%	41%

TABLE V: Evaluation scenarios. The best mAP results are those for which the test and training datasets are from the same dataset.

S(3,3). The results are ranging between 51%-61%. Also at scenarios S(1,3) and S(3,1) and scenarios S(2,3) and S(3,2), the results are also very good.

Rails: The AP results of class "rails" show that all scenarios achieve performance which ranges from a very good to an excellent one. Specifically at scenarios S(1,1), S(2,2) and S(3,3), the performance is ranging from 96% to 99% as expected, because this class has most datapoints compared to other classes. At scenarios S(1,3), S(3,1) and S(3,2) the results are also very good. This can be explained by the fact that the number of datapoints of class "rails" is higher in datasets DS1 and DS3. On the other hand, the surrounding environment that is close to "rails" class objects in the annotated images of dataset DS1 significantly differs from the surrounding environment of this class in the annotated images of dataset DS2. Therefore, the APs at scenarios S(1,2) and S(2,1) are lower but in a relatively satisfactory level.

Terrain: The AP results of class "terrain" range between satisfactory and excellent. The best results are at scenarios where the training and test datasets are from the same dataset. In these cases, the performance ranges from 66% to 93%. Also at scenarios that involve DS1 and DS3 (i.e. S(1,3) and S(3,1)) and at S(3,2) the results are also very good.

The landscape and the various classes of objects we want to trace over and around the rail are very likely to differ greatly in each case. Therefore, training the system on a per case basis produces the best possible results for the classifier performance. Generalizability of classifier is achieved for two main classes, "rails" and "terrain". This is expected because the main features of these classes are common in datasets DS1 and DS2. The classifier performance ranges from satisfactory to excellent level at all evaluation scenarios. This implies that some time can be saved while creating the dedicated dataset for each application case by recording only the parts of a railway which present higher diversity in terms of other railways (e.g. trees and weeds). Indicative examples of evaluated images can be found in Fig. 9 in Appendix I at the end of the paper.

IV. Further Extensions to the Model

A. Faster RCNN Inception ResNet v2 Atrous vs Faster RCNN Inception v2

Critical classes of obstacles, can appear as small scale objects in video frames that will be processed from our classifier (e.g. *"object on rails"*). *Faster RCNN Inception ResNet v2 Atrous* model [31] is a dedicated version of Faster RCNN model suitable for small scale objects that can achieve excellent performance. Therefore we used *Faster RCNN Inception ResNet v2 Atrous* and compared its performance to that of the basic version of the Faster RCNN. Specifically, we compared the generalizability of *Faster RCNN Inception ResNet v2 Atrous* model and its performance to those of the *Faster RCNN Inception v2* model. In the sequel we use the terms *Atrous* and *Inception v2* to refer to the models *Faster RCNN Inception ResNet v2 Atrous* and *Faster RCNN Inception v2* respectively.

In these scenarios we used only datasets DS1 and DS2. DS3 was not used because it is the combination of DS1 and DS2. We follow the same strategy for the evaluation scenarios as in *Inception v2*. We train and test in all possible combinations of DS1 and DS2 and measure the performance based on the AP per label and mAP for four values of IoU i.e. $0.1, 0.2, 0.3$ and 0.4. The classifier models utilizing the *Atrous* were trained with 500,000 training steps with an achieved average total loss 0.06. We trained the *Atrous* in a PC utilizing a GPU NVIDIA Quadro M5000. Training took almost 8 days longer than the *Inception v2* model. Also its memory footprint is much bigger than that of the *Inception v2*. Specifically the trained model based on *Inception v2* is 160MB and the *Atrous* trained model is almost 1GB. Last but not least *Atrous* requires 2.2 sec more than *Inception v2* on average, to process a frame during the classification process.

1) Mean Average Precision Results Comparison: The mAP results for the whole range of IoU in Fig. 7 shows that *Atrous* presents better or similar behaviour with that of the *Inception v2* model. The best results are at the evaluation scenarios where the train and test datasets are from the same dataset. Therefore the generalizability of the classifier is again low.

Compared to the corresponding results of the *Inception v2* model, this classifier has the same mAP for scenario S(2,2), while for other scenarios (S(1,1), S(1,2) and S(2,1)), it gives better results by about 5% on average. Therefore, the classifier performance is improved and it is useful for the application to utilize this model. We examine it in detail at the following subsection, where the average precision for

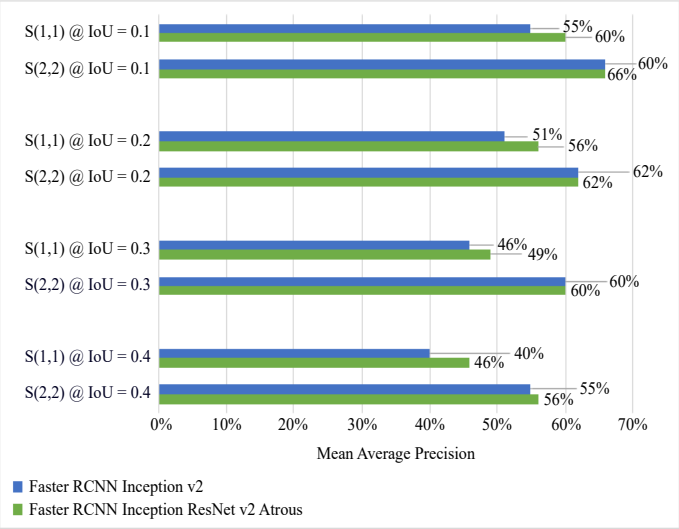

FIG. 7: Overall performance comparison in terms of mAP for different IoU values for the - *Atrous* and *Inception v2* models.

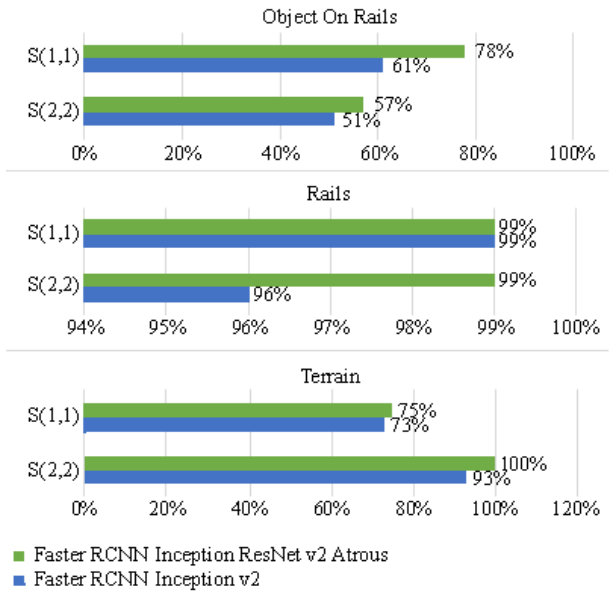

FIG. 8: Comparison of AP for classes: "object on rails", "terrain" and "rails" for different dataset scenarios between *Atrous* and *Inception v2* results. We observe significant improvement especially for the class "object on rails" which includes small-scale objects. This was because of the suitability of *Atrous* for this type of objects.

some indicative classes will be presented for $IoU = 0.1$. This value is the most appropriate one for our case because the objects to be detected are only on the background of the frames that we process.

2) Average Precision per Class Results Comparison: Fig. 8 shows compaative results between the AP results for the *Atrous* and *Inception v2* based classifiers for three indicative main classes: "object on rails", "terrain" and "rails" at $IoU = 0.1$.

Object On Rails: The AP results for class "object on rails" show that the best results are for scenarios where the train and test datasets are from the same dataset (i.e. S(1,1) and S(2,2)). There is significant improvement compared to *Inception v2* model, as the results of the S(1,1) are better by about 20%. The result of the S(2,2) is also better by about 6%.

Rails: The results for the performance of the classifier for the class "rails" are excellent as Fig. 8 shows. For scenarios S(1,1) and S(2,2) we have the best results. There is improvement about 3% in *Atrous* compared to *Inception v2* model for the "rails" class.

Terrain: The classifier has very good performance for the class "terrain" as well. The results in Fig. 8 show that the best performance is observed at scenarios where the training and test datasets come from the same dataset (i.e. S(1,1) and S(2,2)).

Similar to *Inception v2* based classifier, the models are not generalizable. This verifies the need for a dedicated dataset for each case and landscape. *Atrous* has better or equal performance to *Inception v2* in almost all classes. There is definitely improved classification performance for Atrous for many classes and the overall improvement of performance appears on the mAP results. This leads to the conclusion that it would be useful to utilize *Atrous* for the classifier model.

B. Performance Evaluation of the DL Classifier on the UAV using NVIDIA Jetson Nano

One of the main questions is whether it is possible for the classifier to operate on the drone where frames are captured and need to be processed in a streaming fashion. We selected the *NVIDIA Jetson Nano* as the on-board computing system where the DL based classifier will operate. NVIDIA Jetson Nano is an embedded system-on-module (SoM) and developer kit, including an integrated 128-core Maxwell GPU, quad-core ARM A57 64-bit CPU and 4GB LPDDR4 memory. It runs on Linux and provides 472 GFLOPS compute performance with 5-10W of power consumption. We can natively install popular open source Machine Learning (ML) frameworks such as *TensorFlow* and *Keras*, along with frameworks for computer vision and robotics development like *OpenCV*. The space it occupies is 87x58.5x35mm and weights 136gr, therefore it can fit to the specifications of a UAV, for being easily transported, navigated and maneuvered within a railway field.

The UAV on-board video recording subsystem records a video with a resolution 1920x1080 at 25 frames per second (fps). According to our tests, the Jetson Nano takes on average 1.14s to process a frame utilizing the *Inception v2* model based classifier. The *Atrous* model was quite heavy in terms of memory usage, and it was not possible to run it on the Jetson Nano board. Hence, our DL classifier may operate on a computing system with limited resources like the Jetson Nano in order to execute on-board processing scenarios. In fact, given the frame processing time above, it is possible to

use Jetson Nano for processing frames at a video of about one frame per second. This is still good and viable for object detection, since the object or obstacle remains in view of the camera for a few seconds.

V. Conclusion and Future Work

In this paper, we designed, implemented and evaluated a DL based monitoring system for obstacle and vegetation detection along railways through UAV aerial surveying. A prime contribution of our work is the creation of the first dedicated dataset for object detection, that can be further used for the purpose of vegetation management and railway safety. We also used two models of Faster RCNN algorithm, *Inception v2* and *Atrous* to classify objects and vegetation. We trained and evaluated our classifiers and we deduced that the classifier can achieve very good performance when the training dataset is the same as the test dataset.

The main conclusions from the evaluation of the classifier are: *(i)* the development of a dedicated dataset for different railway terrain is recommended and therefore classification knowledge in one terrain cannot be transferred to another. This holds for both the *Inception v2* and *Atrous* models. *(ii)* The *Inception v2* model gives good performance while *Atrous* is even better, especially for small-scale objects.

In this work, the image recognition pipeline was executed and evaluated on the computing system that was employed for the training of the classifier out of the drone system. Next, the trained classifier operated for testing at Jetson Nano on-board the drone. There exist interesting further questions if we assume the existence of an infrastructure, such that we are given the option to move the processing of a frame at different sites e.g. either on-board the drone or at a processor at the edge of the network, e.g. at a base station. In our future work we also plan to further improve our custom dataset. A first option would be to define additional classes for objects that can be confused with the 12 already defined classes. Another option would be to specialize the dataset on very specific cases of terrains e.g. mountainous ones.

We have considered for our next steps the examination of three main operational scenarios. Our first step will be to improve the *on-board* processing by using a computing system such as NVIDIA Jetson Nano. Second, we will consider the *total offload* method, where all video frames are transferred to cloud. And the third option that will be examined is a *hybrid* one, where only selected frames are offloaded to the cloud for complex, resource-demanding tasks. Our final goal is to design and implement an offloading policy to reduce inference delay.

References

[1] Network Rail Vegetation Management Review. [Online]. https://tinyurl.com/y9urf6kl.

[2] Metro Vegetation Management. [Online]. https://tinyurl.com/yc7pxrwp.

[3] Railway Technology Website. "The great tree clearance: does Network Rail need to grow up?". [Online]. https://tinyurl.com/y8drbecb.

[4] M. Karakose, O. Yaman, M. Baygin, K. Murat and E. Akin, "A new computer vision based method for rail track detection and fault diagnosis in railways", *International Journal of Mechanical Engineering and Robotics Research*, 6, 1, pp. 22-17, 2017.

[5] A.K. Singh, A. Swarup, A. Agarwal and D. Singh, "Vision based rail track extraction and monitoring through drone imagery",*ICT Express*, 5, 4, pp. 250-255, 2019.

[6] S. Zheng, X. Chai, X. An and L. Li, "Railway track gauge inspection method based on computer vision", *2012 IEEE International Conference on Mechatronics and Automation*, pp. 1292-1296, 2012.

[7] L. Camargo, E. Resendiz, J. Hart, J. R. Edwards, N. Ahuja and C. Barkan, "Machine vision inspection of railroad track", *NEXTRANS Center (US)*, 2011.

[8] E. Resendiz, J. M. Hart and N. Ahuja, "Automated visual inspection of railroad tracks", *IEEE transactions on intelligent transportation systems*, 14, 2, pp. 751-760, 2013.

[9] M. Singh, S. Singh, J. Jaiswal and J. Hempshall, "Autonomous rail track inspection using vision based system", *2006 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*, pp. 56-59, 2006.

[10] Z. Teng, B. Zhang anf F. Liu, "Railway region detection based on Haar-like features", *Proceedings of International Conference on Internet Multimedia Computing and Service*, pp. 121-124, 2014.

[11] X. Gibert, V. M. Patel and R. Chellappa, "Deep multitask learning for railway track inspection", *IEEE transactions on intelligent transportation systems*, 18, 1, pp. 153-164, 2016.

[12] S. Mittal and D. Rao, "Vision based railway track monitoring using deep learning", arXiv preprint arXiv:1711.06423 [cs], 2017.

[13] C. Aytekin, Y. Rezaeitabar, S. Dogru and I. Ulusoy, "Railway fastener inspection by real-time machine vision", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45, 7, pp. 1101-1107, 2015.

[14] A. James, W. Jie, Y. Xulei, Y. Chenghao, N. B. Ngan, L. Yuxin, S. Yi, V. Chandrasekhar and Z. Zeng, "Tracknet-a deep learning based fault detection for railway track inspection", *2018 International Conference on Intelligent Rail Transportation (ICIRT)*, pp. 1-5, 2018.

[15] Y. Min, B. Xiao, J. Dang, B. Yue and T. Cheng, "Real time detection system for rail surface defects based on machine vision",*EURASIP Journal on Image and Video Processing*, 2018.

[16] UK landscape YouTube video. [Online]. https://tinyurl.com/yabyhye9.

[17] Slovenia landscape YouTube video. [Online]. https://tinyurl.com/yb8cgxsl.

[18] Google's best practices on splitting data. [Online]. https://tinyurl.com/y7yqfhxu.

[19] Tensorflow framework. [Online]. https://www.tensorflow.org/.

[20] MSCOCO dataset format. [Online]. http://cocodataset.org/.

[21] FFMPEG software. [Online]. https://www.ffmpeg.org/.

[22] Labelme software. [Online]. https://github.com/wkentaro/labelme.

[23] Adobe Photoshop software. [Online]. https://www.adobe.com/.

[24] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *Advances in neural information processing systems,* pp.91-99, 2015.

[25] R. Girshick, "Fast R-CNN", *Proceedings of the IEEE international conference on computer vision,* pp.1440-1448, 2015.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556,* 2014.

[27] W. Liu, D. Anguelov, D Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg, "Ssd: Single shot multibox detector", *European conference on computer vision*, pp.21-37, 2016.

[28] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection", *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp.779-788, 2016.

[29] L. Bottou, "Large-scale machine learning with stochastic gradient descent", *Proceedings of COMPSTAT'2010,* pp. 177-186, 2010.

[30] Tensorflow Model Zoo. [Online]. https://tinyurl.com/yd7ghyxt.

[31] T. Guan and H. Zhu, "Atrous faster R-CNN for small scale object detection", *2017 2nd International Conference on Multimedia and Image Processing (ICMIP)*, pp.16-21, 2017.

APPENDIX I

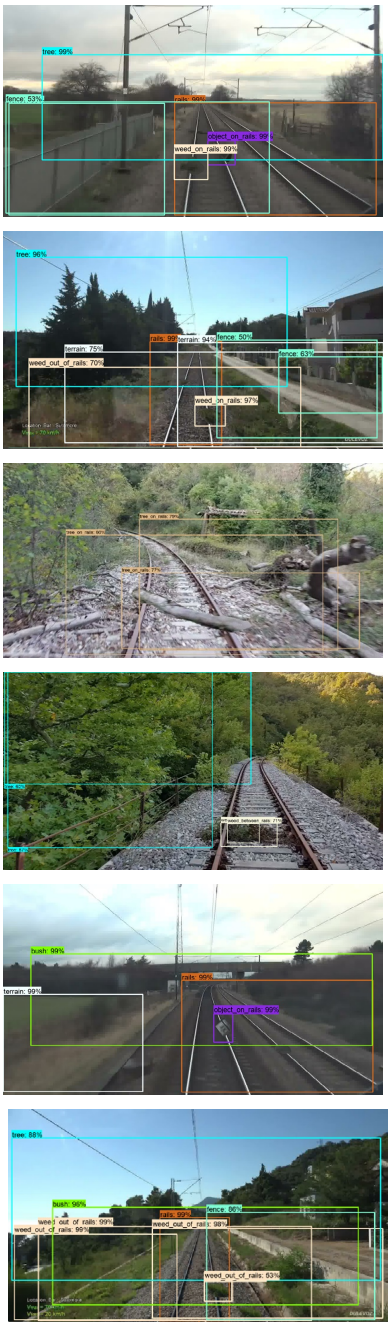

FIG. 9: Examples of evaluated images. We observe the bounding boxes for each detected class and corresponding prediction of existence into this box as percentage. The bounding boxes especially for small objects like in the class "weed on rails" or "object on rails" are tightly and well fitted around the object.

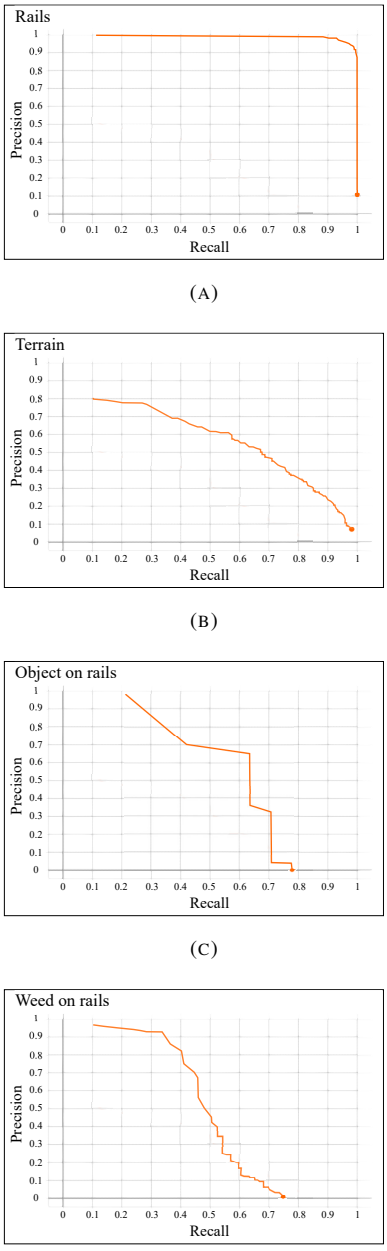

FIG. 10: Indicative Precision-Recall (PR) curves for classes: (A) "rails", (B) "terrain", (C) "object on rails" and (D) "weed on rails". The PR curves of a perfect classifier shows a combination of two curves – from the top left corner (0.0, 1.0) to the top right corner (1.0, 1.0) and further down to the end point. The PR curve for class "rails" is very close to a perfect curve and has the most area under its curve than other classes. Similar trends are observed for other classes.