Improving Learning Effectiveness For Object Detection and Classification in Cluttered Backgrounds

Vinorth Varatharasan[1], Hyo-Sang Shin[1], Antonios Tsourdos[1] and Nick Colosimo [2]

***Abstract*— Usually, Neural Networks models are trained with a large dataset of images in homogeneous backgrounds. The issue is that the performance of the network models trained could be significantly degraded in a complex and heterogeneous environment. To mitigate the issue, this paper develops a framework that permits to autonomously generate a training dataset in heterogeneous cluttered backgrounds. It is clear that the learning effectiveness of the proposed framework should be improved in complex and heterogeneous environments, compared with the ones with the typical dataset. In our framework, a state-of-the-art image segmentation technique called DeepLab is used to extract objects of interest from a picture and Chroma-key technique is then used to merge the extracted objects of interest into specific heterogeneous backgrounds. The performance of the proposed framework is investigated through empirical tests and compared with that of the model trained with the COCO dataset. The results show that the proposed framework outperforms the model compared. This implies that the learning effectiveness of the framework developed is superior to the models with the typical dataset.**

I. INTRODUCTION

There have been growing attention on the exploitation of potential benefits of Unmanned aerial vehicles (UAVs). Their potential applications under discussion are extensive, e.g., surveillance, delivery, defence, agriculture, and so on. For instance, if a small UAV is considered, it can be used in civil applications such as delivery or supply, or in defence applications such as military resupply or sacrificial weapons; therefore a trade-off between a low cost and good performance is needed. Many sensors exist, such as lidars, radars, electro-optical cameras, etc. The major challenge of the latter is that there is not direct range information since there are difficulties to differentiate between a close and a small object, or a far and a big object, because the angle subtended is the same.

This is the reason behind the use of neural networks in parallel. These tools are powerful for computer vision applications and can be very useful in sense-and-avoid or other applications. For example, if a particular kind of object is classified, stadiametric rangefinding and simple trigonometry could be used to find the depth of an object.

The close and frequent proximity to urban obstacles and numerous other airspace users (UAV in particular) presents a new level of challenge. However, there are also other applications such as situational awareness, in which robust detection algorithms have to be implemented.

[1] School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield MK43 0AL, UK (email: vinorth.varatharasan@gmail.com)

[2] BAE Systems Warton Aerodrome, Warton, Preston, PR4 1AX

Last but not least, background clutter easily confuses the machine. Indeed, when the hardware is running object detection and tracking algorithms, the results can be false because of the similarity between the object of interest (target) and the background.

Existing state-of-the-art object detection and tracking techniques provide great performances when observing the objects of interest with homogeneous backgrounds. However, the performance dramatically degrades when observing the objects of interest with cluttered backgrounds. The main aims are thus to have good performances in any background, and for that, an innovative method of training is introduced. Therefore, the following objectives were considered:

- Improve detection and classification probability while minimising false alarm rate which is a function of the background scene clutter.
- Verify and validate the proposed methodology by using existing detection and classification accuracy metrics, by comparing the performances in heterogeneous backgrounds.

The rationale behind the poor performance of the state-of-the-art object detection techniques in cluttered backgrounds is the lack of the use of heterogeneous backgrounds in the training set.

Thus, instead of using homogeneous backgrounds which are typically used in training, heterogeneous backgrounds are introduced in the training dataset by creating an artificial data using Deep Learning techniques to improve the learning efficiency. This proposed framework is automated to create a large training dataset, and hence help achieve the requirements by obtaining accuracy improvements in object detection.

Nowadays, more and more background replacement tools using Artificial Intelligence are being implemented in various sectors. For instance, Microsoft develops this for Skype for Business [1], and Google AI tries to remove background from YouTube videos in real-time [2] using Chroma-keying. Many other open-source tools are also available to the public.

Some research has also been done to obtain better performance in cluttered scenes (Feature extraction [3] [4] [5], Filtering [6], and so on). Furthermore, data augmentation is widely used today in order to improve the learning effectiveness, for example by changing the lightning, the rotation, the size, the variety of backgrounds (e.g. snow, rain, etc.) of the images. A similar approach to our proposed methodology, explained in Chapter II, was introduced in [7].

This study also tried to improve training effectiveness for object classification in cluttered environments, but without using Deep Learning techniques. In fact, background removal must have been done manually and spectral texture-based features were used for back-propagation. Compared to the former approaches, the use of AI techniques generates a dataset with better accuracy and in a very limited time.

The paper starts with the explanation of the proposed framework, which is split into two essential parts: an object segmentation technique performing a background removal and a chroma-keying process conducting a background replacement. These two techniques are used to generate autonomously a dataset of images in cluttered backgrounds which is then used to train the object detection model. The performance assessment is then considered to validate the proposed approach by using numerical simulations, with the comparison between a model trained with the COCO dataset and the model trained with the dataset generated by the proposed framework. Finally, the main outcomes are summarised, followed by critics and proposals.

II. PROPOSED METHODOLOGY

The integration of two different techniques is used to respect the aims and objectives set in the Introduction:

- An image segmentation technique called DeepLab.
- The Chroma-keying technique.

In other words, these techniques are used to obtain an unreal dataset of images with heterogeneous cluttered backgrounds, which is then used to train the Neural Network model. By implementing this strategy, the learning (or training) effectiveness improves and therefore, the detection and classification accuracy gets better in heterogeneous backgrounds and by extension, in any background.

For example, the Neural Network is trained on a large dataset in different cluttered backgrounds (e.g. 2000 images $\times$ 5 cluttered backgrounds = dataset of 10000 images), which is also time-saving because only 2000 images need to be labelled as ground truths for 10000 images (5 times faster).

A. Image segmentation

On the one hand, the object detection generates bounding boxes around the objects of an image and classifies each one of these boxes into a class. But on the other hand, the object segmentation classifies each pixel of the image into a class, which permits to locate the detected objects/instances more precisely.

There are two types of image segmentation:

- Semantic segmentation.
- Instance segmentation.

Many state-of-the-art semantic algorithms exist:

- Fully convolutional networks [9].
- Encoder-decoder networks: SegNet [10], U-Net [11], DeconvNet [12].
- Dilated networks: DilatedNet [13], PSPNet [14], DeepLab (v1 [15], v2 [16], v3 [17], v3+ [18]).

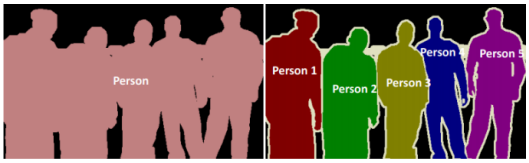


Fig. 1. Semantic segmentation (left) and Instance segmentation (right) [8]

Same for region-based instance segmentation algorithms:

- Multi-task Network Cascade (MNC [19])
- Fully Convolutional Instance-aware Semantic Segmentation (FCIS [20])
- Mask-RCNN [21]

Some semantic segmentation techniques may be real-time; however, instance segmentation is much more powerful and interesting for their accuracy performances in extracting the maximum details from the objects of interest.

Many state-of-the-art instance segmentation methods exist, and similar techniques always get improved with time. The average precision (AP) is usually used as an accuracy metric, which is explained in Section III-B.

TABLE I
COCO CHALLENGE RESULTS FOR INSTANCE SEGMENTATION TECHNIQUES [21]

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
MNC	ResNet-101-C4	24.6	44.3	24.8	4.7	25.9	43.6
FCIS +OHEM	ResNet-101-C5-dilated	29.2	49.5	-	7.1	31.3	50.0
FCIS+++ +OHEM	ResNet-101-C5-dilated	33.6	54.5	-	-	-	-
Mask-RCNN	ResNet-101-C4	33.1	54.9	34.8	12.1	35.6	51.1
Mask-RCNN	ResNet-101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
Mask-RCNN	ResNeXt-101-FPN	**37.1**	**60.0**	**39.4**	**16.9**	**39.9**	**53.5**

Thus, without tricks, Mask-RCNN outperforms all existing, single-model entries on every task, even the COCO challenge winners of 2015 and 2016. As a result, **Mask-RCNN** can be used for object segmentation purposes.

However, since only the background needs to be removed, if we only obtain the shape of the desired objects like in Figure 1, we can extract these objects' shapes with semantic segmentation techniques which present better accuracy than Mask-RCNN for these goals, as described in Table II.

TABLE II
PERFORMANCE (*mean IoU*) OF SOME IMAGE SEGMENTATION TECHNIQUES

Model name	*mIoU*
FCN	62.2
DeepLabv2-CRF	79.7
PSPNet	85.4
DeepLabv3	87.3
DeepLabv3+	89.0
Mask-RCNN	**74.4**

Hence, semantic techniques such as **DeepLab** have much better results than Mask-RCNN and can be used to meet the predefined requirements. Therefore, we use DeepLabv3 for image segmentation.

First of all, a semantic image segmentation technique with DeepLabv3 [22] in TensorFlow (Python) is introduced. It is used to remove the background of every image from the training dataset, which is then replaced by a green background, as illustrated in Figure 2.

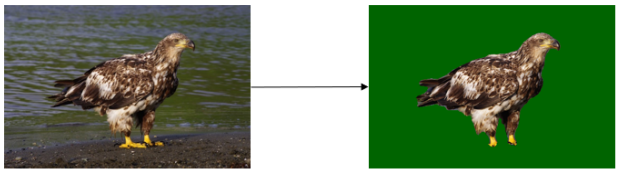


Fig. 2. Object segmentation for background replacement

The background extraction algorithm is implemented as follows:

1) The latest version of the pre-trained DeepLab model is loaded. Two models are proposed: the MobileNetv2 [23] model and the Xception [24] model. The MobileNetv2 model is much faster but less accurate than the Xception model. MobileNetv2 is a fast network structure intended for mobile devices, whereas Xception is a more robust network structure designed for server-side deployment. Thus, the latter is used since accuracy is essential for our application.
2) After the model is loaded, the inference is run on all images in a folder named *input-background-removal* so that all classes (objects) defined in the pre-trained model are segmented.
3) Finally, a distinction between the segmented objects and the background is made. Therefore, the background can be easily removed and replaced by a green screen background by defining the RGB pixels to $[0, 100, 0]$.
4) All the images with the green screen background are outputted in a folder named *input-with-green-screen-background* so it can be used for the chroma-key technique in Section II-B.

B. Chroma-key

Secondly, a Chroma-keying technique is used to remove the green background from the training dataset (e.g. the output of the first background replacement in Figure 2) to replace it by any cluttered background as illustrated in Figure 3.

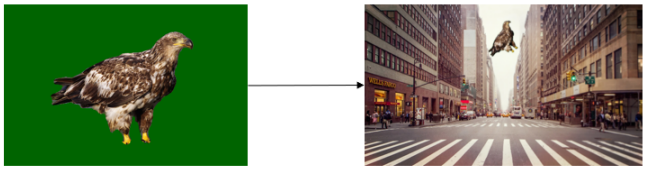


Fig. 3. Chroma-key technique

The chroma-keying algorithm [25] uses the skimage [26] Python library to realise many imaging operations, such as:

- Green background removal from an image.
- Merging of several pictures into one, which also permits necessary rotations, translations and scale operations of the objects of interest.
- Filtering of the image.
- Storage of the outputs.

1) Green background removal: The green background is removed from the input images depending on a predefined threshold.

The *NumPy* [27] library enables to use an array of four-channel image: $RGB\alpha$.

Then, the ratio of the red/green/blue channels based on the max-bright of the pixel is obtained by dividing the RGB pixels by the norm factor (therefore 255):

$$colour_{ratio} = \frac{colour_{pixel}}{255} \tag{1}$$

where the colour is red, blue or green.

Dark pixels are almost equal to zero. Consequently, when calculating the red/blue vs green ratios, we can obtain small negative values for dark pixels (the "/" sign is here used as "or"). An additional parameter of 0.2 is added to these values to avoid this issue.

$$(red/blue)vs(green) \equiv (red/blue)_{ratio} - green_{ratio} + 0.2 \tag{2}$$

The additional parameter was manually tuned to obtain better performances for the green colour removal.

In the following pictures (from Figure 6 to Figure 11), we can see a comparison of the green screen background removal outputs with this additional parameter varying between 0.1 and 0.3 with Figure 4 and Figure 5 as inputs with a green screen.

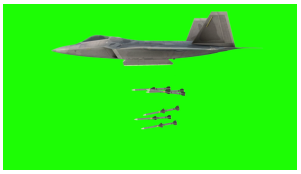


Fig. 4. Input 1 for green screen removal

Fig. 5. Input 2 for green screen removal

Consequently, when the additional parameter is below 0.2, some information of the plane is missing. Nevertheless, with values above 0.2, some green pixels are not correctly removed, and hence the choice of the additional parameter with a value of 0.2.

The remaining negative values (even after adding 0.2 to the parameter) become zero in Equation (2).

After that, the red/blue vs green ratios need to be combined to set an α-layer:

Fig. 6. Green screen removal with an additional parameter 0.1 (example 1)

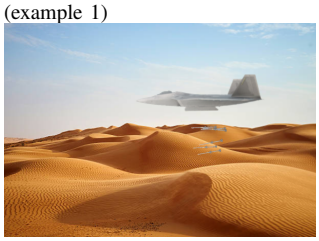


Fig. 7. Green screen removal with an additional parameter 0.1 (example 2)

Fig. 8. Green screen removal with an additional parameter 0.2 (example 1)

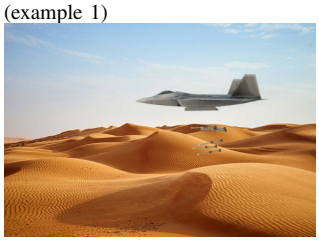


Fig. 9. Green screen removal with an additional parameter 0.2 (example 2)

Fig. 10. Green screen removal with an additional parameter 0.3 (example 1)

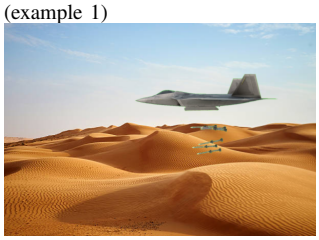
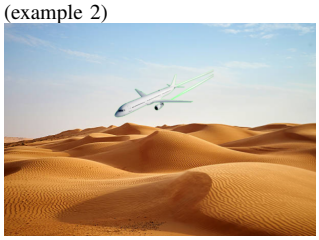


Fig. 11. Green screen removal with an additional parameter 0.3 (example 2)



$$\alpha \equiv ((blue)vs(green) + (red)vs(green)) * 255 \quad (3)$$

Then the values of alpha above 50 become the norm factor value, which is 255.

Finally, the fourth value α of the NumPy array $RGB\alpha$ is obtained and thus, the new image after setting these tuned parameters is the image without the green background.

2) Blending: After removing the green background, the integration of the two following images must be performed:

- The image on the foreground, containing the objects of interest (output of the image segmentation), needs to be an image with an alpha layer (as calculated in the previous Subsection II-B.1).
- The desired background, which is an inhomogeneous cluttered background.

Furthermore, basic rotation and scaling operations can be done (randomly or in a predefined way) thanks to the *skimage* Python toolkit.

3) Channel adjustment: In order to emphasise colours or other features in an image, a curve is applied to remap the image tonality.

It can be used to each channel individually in an image or to all channels together. The first option can be employed to stress the colour.

The sigmoid function is used to make adjust-curves:

$$S(x) = \frac{1}{1 + \mathrm{e}^{-x}} = \frac{\mathrm{e}^{x}}{\mathrm{e}^{x} + 1} \quad (4)$$

where x is a NumPy array and e^{x} is the NumPy exponential function.

4) Filtering: First, the left side of the image is blurred with a Gaussian filter [28].

Then, the α-values need to be decreased gradually from right to left.

After that, the blurred image and the background are merged into one image to create an appearance of partial or full transparency. This method is called *alpha compositing* [29]. The final output of this technique, called the *composite*, is obtained after rendering image elements in separate passes and then combining the resulting multiple 2D images into a single one.

At last, the final outputs of this whole process are saved in a file named *output*, ready to be part of the training dataset.

C. Seg-CK

The two techniques described in Section II-A and Section II-B, the image segmentation and Chroma-key, are then integrated as one technique that we named **Seg-CK**. Its architecture is illustrated in Figure 12.

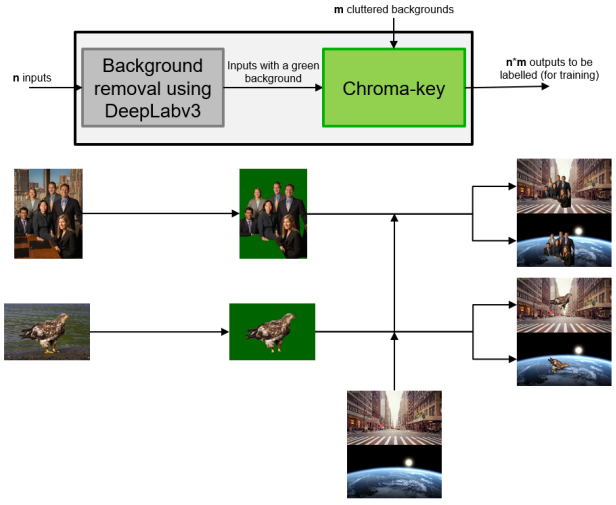


Fig. 12. Seg-CK architecture

Thus, n image inputs first go through the DeepLab background removal process to obtain n outputs with a green background.

Then, the Chroma-key technique takes two inputs:

- The n outputs of the background removal block with a green screen background.
- The m relevant heterogeneous backgrounds that can be useful for applications such as fully-autonomous UAV

(e.g. backgrounds with a lot of buildings, cars, street furniture, and so on).

As a result, we get $n \times m$ outputs with the n images containing the objects of interest that are most likely to be found during a UAV trajectory (e.g. birds, drones, buildings, etc.) in the m cluttered backgrounds.

D. Model training

1) Labelling: The $n \times m$ outputs are labelled using a GitHub repository named *LabelImg* [30].

It is a graphical image annotation tool using Qt for the graphical interface and written in Python.

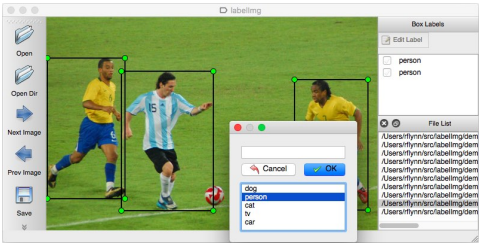


Fig. 13. LabelImg for labelling images [30]

The output annotations can be saved in two different formats:

- XML files in PASCAL VOC format, used by ImageNet [31].
- TXT files supported by YOLO [32].

YOLOv3 [33] in TensorFlow is used to train the CNN model and to test the performances, hence the TXT format is used. It is a text file containing one or several lines with this specific order: *class* x_{min} y_{min} x_{max} y_{max}.

For example, if we label a picture containing two classes (bird and aeroplane), we can have:

bird 41 224 224 341

aeroplane 295 80 583 294

2) Training: Two different training dataset is compared for the performance assessment purposes:

- The COCO dataset with only its aeroplane class.
- The artificial dataset generated from the proposed framework, also with only one aeroplane class.

3) Training of the COCO dataset: After training the model with only the aeroplane class from the COCO dataset in darknet [34], we just needed to convert the darknet weights to a TensorFlow checkpoint. These convolution weights, with 53 convolutional layers, and therefore called Darknet-53, are trained on ImageNet [31].

Then, a Python weight converter was used to convert this *yolov3.weights* the darknet weights [34] to a TensorFlow checkpoint file named *yolov3.ckpt*.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig. 14. Darknet-53 [33]

4) Training of the dataset created with the Seg-CK framework: We suppose that we created 10000 images with the Seg-CK method (2000 images of an aeroplane in 5 different cluttered backgrounds) and that these pictures have been labelled. Also, we wanted to have the same proportion of planes and drones as in the aeroplane class from the COCO dataset.

In order to train a YOLOv3 model with the specific dataset, three main files need to be created:

1) *aeroplane.data*
2) *aeroplane.names*
3) *aeroplane.cfg*

Firstly, in the *data* file, the details that need to be mentioned are:

- The number of classes.
- The train set file.
- The validation set file.
- The file that contains the names of the classes we want to train and hence to detect.
- The folder where the yolo weights file is stored.

Consequently, we created the following *data* file:

```
classes = 1
train = aeroplane-train.txt
valid = aeroplane-test.txt
names = aeroplane.names
backup = backup/
```

Fig. 15. *aeroplane.data* file

Then, as its name suggests, *aeroplane.names* contains all the names of the classes.

```
aeroplane
```

Fig. 16. *aeroplane.names* file

Last but not least, many vital parameters need to be defined in *aeroplane.cfg*:

- $batch = 48$, which means that for each training step, 48 images are used.

- $subdivisions = 16$, which is used to decrease the GPU VRAM (Graphics Processing Unit Video Random Access Memory) requirements by dividing the batch by 16.
- $classes = 1$, because the only category we train is the aeroplane class.
- $filters = (classes + 5) \times 3$, therefore $filters = 18$.

The batch and subdivisions are tuned parameters to get better performance.

III. PERFORMANCE ASSESSMENT

After the Neural Network model was trained, accuracy metrics such as the *mean average precision* (mAP) are used to assess the performances of:

- The detection accuracy.
- The classification accuracy.

We did not conduct an assessment with planes/drones or backgrounds pictures that were trained in the aeroplane class.

However, we chose to train 2000 different images from the aeroplane class in five different cluttered backgrounds ($2000 \times 5 = 10000$ pictures). Different heterogeneous backgrounds were also used (e.g. complex urban environments).

Nevertheless, as explained in the three following sections, we still get excellent performances in pictures and videos of an aeroplane in other kinds of backgrounds as shown in Figure 17 and Figure 27 (e.g. forest, very cloudy sky, buildings, space).

A. Detection

In this section, detection accuracy is compared between two different models:

- The model trained with COCO dataset.
- The model trained with the artificial dataset created thanks to the proposed framework.

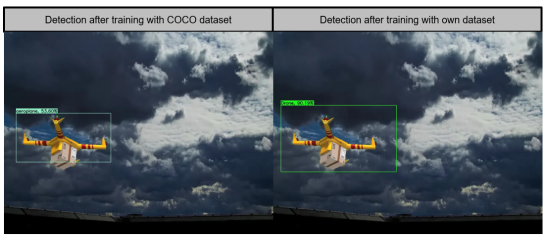


Fig. 17. Frame captures of a video applying YOLOv3 in a heterogeneous environment

In order to conduct a performance assessment in highly cluttered backgrounds, existing videos of aircraft in heterogeneous environments were taken (e.g. from YouTube).

With the chosen accuracy metric (mAP), only pictures can be evaluated. Hence, the following steps were done:

1) It is necessary to have two videos: an original video and the same video outputted from YOLO.
2) Then these two videos need to be divided into the same number of frames, for example, 100 frames.
3) From the frames of the original video, ground truths are created.
4) Then, each frame of the ground truth and YOLO result is compared one by one with the mAP metric.

With three videos of aeroplane, which can be split into 300 frames for a total 27 seconds of videos, in heterogeneous backgrounds (such as the one in Figure 17), the mAP was approximately 38% for the model trained with the COCO dataset, whereas the average for the model trained with the proposed framework was 86%.

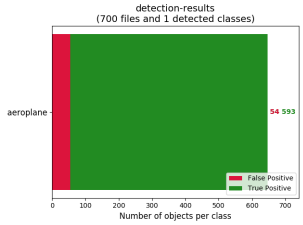
These values were auspicious, and thus, classification accuracy needed to be evaluated with a more diversified validation dataset to confirm this analysis.

B. Classification

In this section, 700 images of aeroplane are tested in different heterogeneous cluttered backgrounds (images containing planes/drones and backgrounds utterly different from the training set).

Fig. 18. Detection results in complex environments (model trained with the COCO dataset)

Fig. 19. Detection results in complex environments (model trained with the proposed framework)



Here, on the one hand, the model trained with the COCO dataset's aeroplane class gives a mean average precision of $\frac{96}{700} = 13.71\%$, which is an appalling performance. However, the proposed methodology gives a mean average precision of 83.21%. In fact, after training the Neural Network model with the proposed framework, YOLOv3 detected 647 times out of 700 that an aeroplane was present, but only 593 times of them were true positives (which means that the prediction was above the predefined threshold IoU of 0.5). The 54 others are false positives (the IoU is smaller than 0.5, or a bounding box is duplicated).

Intersection over Union (IoU) represents the overlapping area over the combined area of two bounding boxes.

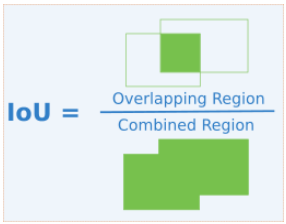


Fig. 20. Intersection over Union [35]

To obtain the IoU, two types of bounding boxes are needed:

- The *predicted bounding boxes* from the YOLO model.
- The *ground-truth bounding boxes*, which are usually created by the user for comparison means (e.g. by labelling images).

Therefore, an IoU threshold of 0.5 was predefined since a similar threshold was used for the official COCO [36] dataset accuracy testing. Thus, a true positive is when a label of a picture has an IoU greater than 50% when comparing the predicted bounding box and the ground-truth bounding box.

Fig. 21. Mean average precision with complex test dataset (model trained with the COCO dataset)

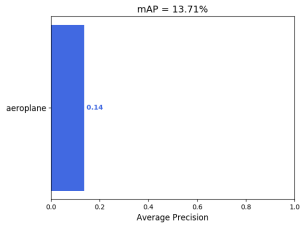
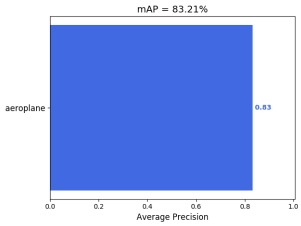


Fig. 22. Mean average precision with complex test dataset (model trained with the proposed framework)



The average precision is computed by:

$$AP_k = \frac{1}{G_{TP}} \sum_{i=1}^{k} \frac{TP_{seen}}{i} \tag{5}$$

where G_{TP} is the total number of ground-truth positives (labelled-as-positive data), TP_{seen} is the number of true positives seen and AP_k is the k^{th} picture's average precision.

Finally, the mean of all the Average Precision is considered as the final accuracy metric (mAP):

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{6}$$

This metric was chosen mostly because it is the principal metric when conducting a performance assessment of a state-of-the-art object detection technique.

Fig. 23. Log-average miss rate with complex environments (model trained with the COCO dataset)

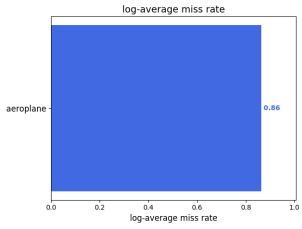
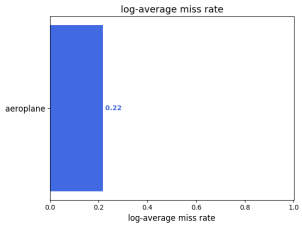


Fig. 24. Log-average miss rate with complex environments (model trained with the proposed framework)



The definition of miss-rate is:

$$MR = \frac{FN}{TP + FN} \tag{7}$$

where MR is the miss-rate, TP is the number of true positives and FN is the number of false negatives (which means the IoU is higher than 0.5, but with a wrong classification).

Fig. 25. Average precision score, micro-averaged over the aeroplane class (model trained with the COCO dataset and tested with complex images)

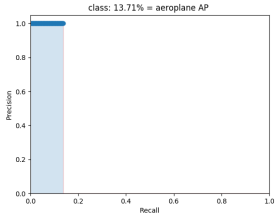
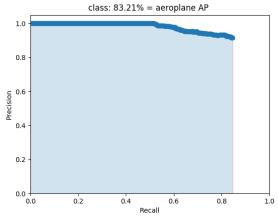


Fig. 26. Average precision score, micro-averaged over the aeroplane class (model trained with the proposed framework and tested with complex images)



Thus, the performance difference is huge. The model trained with the proposed framework gives an improvement of $\frac{83.21}{13.71} = \mathbf{607\%}$ compared to the model trained with the COCO dataset when performing an object detection technique (YOLO) in cluttered backgrounds, for one trained class.

Furthermore, COCO dataset is an excellent object detection dataset with 80 classes, 80000 training images and 40000 validation images. However, it has 3083 for the aeroplane class. In terms of comparison, the proposed methodology has 10000 training images for the aeroplane class (planes and drones with the same proportion as the COCO dataset), including 2000 different images in five cluttered backgrounds, and 700 validation images.

Figure 27 contains three examples of the same UAV in three different heterogeneous backgrounds during a one-by-one detection/classification accuracy assessment in different positions:

Fig. 27. Classification accuracy assessment (mAP) in three different cluttered backgrounds (ground-truth box in blue and YOLOv3 detection result box with the proposed method of training in green) by performing IoU

IV. CONCLUSIONS

Accuracy is an essential requirement in computer vision applications, however background clutter in real-world conditions degrades accuracy performances. In this paper, we focused on improving accuracy when performing a real-time object detection technique called YOLO in highly cluttered

environments. For that, an innovative framework of generating an artificial training dataset for neural network models was implemented. It is based on two open-source codes: a state-of-the-art semantic segmentation model DeepLab to extract the objects of interest from the selected images, and Chroma-key, a technique which merges the extracted objects into predefined heterogeneous background. The resulting framework was called Seg-CK and permitted the training process to be more efficient. Moreover, a model trained with the proposed framework was shown to be six times more accurate in cluttered backgrounds than models trained with existing images (e.g. COCO dataset).

In general, the proposed methodology is shown to be an up-and-coming tool in considered contexts. Therefore, further research on this kind of training method is recommended. For instance, the number of different cluttered backgrounds in the training set is still tunable. Also, further research can be done on GAN (Generative Adversarial Network) in order to generate the backgrounds since it is a potent and promising tool to generate data from scratch.

REFERENCES

[1] Esat Dedezade. Microsoft teams new customized background feature hides distractions. https://news.microsoft.com/.

[2] Valentin Bazarevsky and Andrei Tkachenka. Mobile real-time video segmentation. https://ai.googleblog.com/2018/03/mobile-real-time-video-segmentation.html.

[3] A. S. Ben-Musa, S. K. Singh, and P. Agrawal. Object detection and recognition in cluttered scene using harris corner detection. pages 181–184, July 2014.

[4] A. Bouganis and M. Shanahan. Flexible object recognition in cluttered scenes using relative point distribution models. pages 1–5, December 2008.

[5] Karol Hausman, Christian Bersch, Dejan Pangercic, Sarah Osentoski, Zoltan-Csaba Marton, and Michael Beetz. Segmentation of cluttered scenes through interactive perception. 2012.

[6] A. Shahbaz and K. Jo. Optimal background modeling for cluttered scenes. pages 5240–5244, October 2017.

[7] Bagavathi Nagarajan and Prof P Balasubramanie. Neural classifier for object classification with cluttered background using spectral texture based features. *Journal of Artificial Intelligence*, 1, February 2008.

[8] Computer vision tutorial: A step-by-step introduction to image segmentation techniques. https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentati.

[9] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[10] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[12] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[13] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, 2015.

[14] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

[15] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014.

[16] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[18] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018.

[19] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. *CoRR*, abs/1512.04412, 2015.

[20] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. *CoRR*, abs/1611.07709, 2016.

[21] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[22] Deeplab: Deep labelling for semantic image segmentation. https://github.com/tensorflow/models/tree/master/research/deeplab.

[23] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[24] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[25] Chroma-key. https://github.com/RaulSanchezVazquez/chroma-key-composition.

[26] Stefan van der Walt, Johannes Schonberger, Juan Nunez-Iglesias, Francois Boulogne, Joshua Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. Scikit-image: Image processing in python. *PeerJ*, 2, July 2014.

[27] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: a structure for efficient numerical computation. *CoRR*, abs/1102.1523, 2011.

[28] Estevao Gedraite and M Hadad. Investigation on the effect of a gaussian blur in image filtering and segmentation. pages 393–396, January 2011.

[29] Thomas K. Porter and Tom Duff. Compositing digital images. 1984.

[30] Labelimg. https://github.com/tzutalin/labelImg.

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. 2009.

[32] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[33] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[34] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[35] Intersection over union (iou) for object detection. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou.

[36] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. pages 740–755, 2014.