Classification of Objects in Images using Neural Networks

Elliot Wyman,
Henrik Van Der Horst,
Dulith M Fernando
Department of Decision Sciences
HEC Montreal,
Montreal, QC H3T 2A7

Abstract

The aim of this project was to classify objects in images using an existing neural network model. The focus of this project will be on detecting the mentioned 5 classes in the images. In this project, a multitude of deep learning techniques will be utilized to fine-tune the pre-trained model to analyze and recognize objects in a range of settings. This was achieved by fine-tuning the architecture and weights of the "ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8" pretrained model. In fact, the model was five-tuned during five iterations, with variations in input image size, batch size, and training length. Results were measured in terms of average precision and recall for detections with Intersection over Union (IoU) between 0.5 and 0.95. The version with input images of size 352x288, batch size of 64, number of steps trained of 43k, and medium object size showed the highest average precision and recall. The conclusion was that the precision and recall were satisfactory for larger objects, but the performance for small objects was relatively poor and affected overall metrics.

1 Introduction

The rapid growth of Big Data has resulted in a proliferation of data-driven initiatives being conducted on a global scale. One such initiative, on the civic-level is the move towards becoming “smart cities”, where data is leveraged to optimize processes such as traffic management and construction. Furthermore, to promote individuals and organizations to create such innovative projects, many cities have established open data platforms. An important tool in this process is the automation of surveillance camera video analysis, which can process large volumes of video data in real-time and assist in addressing specific urban challenges. The city of Montreal depends on 531 surveillance cameras placed at the most significant intersections in the city to monitor events occurring at these locations at all times. While this system is useful, it does not facilitate easy tracking of significant incidents taking place in these areas. Manually recording every activity would be a time-consuming task. To make this system scalable and enable automatic recording of all events that occur at an intersection on any given day (such as congestion, accidents, and construction work), the solution could be to create a Machine Learning Model able to identify all of these events. The first step towards this model is to ensure that the system can recognize objects in the videos before being able to recognize specific events.

For this project, the first step was to leverage public data available on the open data platform of the city of Montreal. This was done by using the ‘Annotated Images of Traffic Cameras’ dataset, containing 10,000 annotated images of Montreal traffic intersections, and providing a unique resource for researchers interested in studying transportation patterns and behavior in the city. Specifically, this dataset provides detailed annotations of vehicle and pedestrian positions, as well as construction zones on surveillance camera images.

This project intends using machine learning algorithms on this public dataset to be able to detect and classify objects belonging to 5 classes on the surveillance camera images. Namely, vehicles, pedestrians, buses, cyclists, and construction sites. Detecting the objects is the first step on the long-term project of identifying them in real-time, be able to effectively track their movements and derive meaningful insights that can improve traffic flow and alleviate congestion in the city. To get real-time objects detected accurately and quickly, predictions can be made after having a trained model that has good performance on unseen data.

2 Literature Review

2.1 Related Work

An article published by Grondin (2021), talks about a similar project where the author developed an object detection solution for video feeds. Grondin utilized the same 'Annotated Images of Traffic Cameras' dataset from the City of Montreal which comprised of 10,000 images with their associated objected detection annotations in XML format. To help provide more training examples, an additional data set with very similar angles of view and with similar resolutions was used. YOLOv5 model was selected as the foundation for this project. Furthermore, the Single Shot Multibox Detector (SSD) was selected as a baseline model to compare with YOLOv5. The best and final models were evaluated on the test set. The table below shows the final performance on the data of intersections seen during training (in-domain) and the data of intersections not seen during training (out-domain).

Table 1: Final model performance on in-domain and out-domain test sets

Model Name	**mAP** 0.5 IoU	**Latency** **CPU | GPU** (ms)
YOLOv5-S-512 (in-domain)	0.753	167.1 | 4.2 ms
YOLOv5-S-512 (out-domain)	0.645	166.6 | 4.2 ms
YOLOv5-X-512 (in-domain)	0.820	1398.0 | 16.6 ms
YOLOv5-X-512 (out-domain)	0.692	1398.0 | 17.1 ms

As expected, the performance on data of intersections seen during training is better than the data of intersections not seen during training. The best model performance was on the YOLOv5-X model that had an input of images that were of the size 512x512. This model had a mean average precision (mAP) of 0.82. Grondin stated that, although the performance deteriorated on unseen intersections – the quality of the detection remains excellent to the human eye. Meaning that, while the algorithm may not have been able to generalize well to new intersections, it was still able to identify objects with a high degree of accuracy, such that a human observer would still consider the detections to be of high quality.

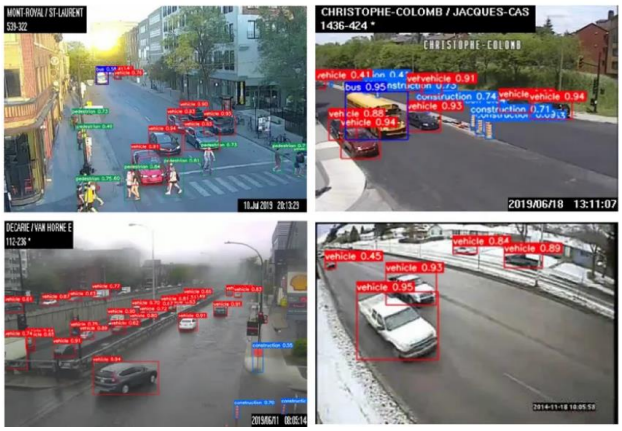

Figure 1. Output from Grondin's pretrained model

2.2 YOLOv5 Model Architecture

According to Snegireva & Perkova (2021) YOLOv5 uses a deep convolutional neural network to detect objects in an image, by predicting bounding boxes and class probabilities for the objects present in the image. The whole architecture is said to be divided into three main parts: feature extractor, detector, and classifier. When a new image enters the network, it passes through the feature extractor to extract feature maps of different scales. Then the feature maps are fed into the detector to obtain information about the location of bounding boxes and the belonging of the found object to one of the defined classes is determined by the classifier. In comparison to SSD, the YOLOv5 model has fewer layers and hence a simpler architecture than SSD.

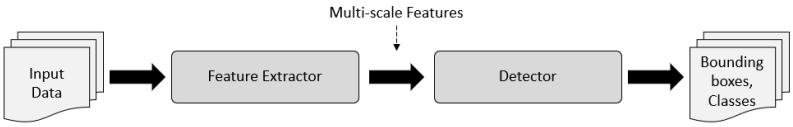

Figure 2. General Architecture of the YOLOv5 Architecture

2.3 Other Pre-trained Models used for Object Detection

2.3.1 R-CNN (Region-based Convolutional Neural Networks)

As per Das et al. (2020), R-CNN architecture contains three steps in which the first step is to take the input image and scan it for probable inputs using an algorithm called selective search. This generates ~2,000 regions. Thereafter, a convolutional neural network (CNN) is run on top of each region. Lastly, each CNN output into (i) a support vector machine (SVM) and (ii) a linear regressor. The reason being, SVM classifies the region of object in an image and linear regressor tightens the bounding box of the detected object.

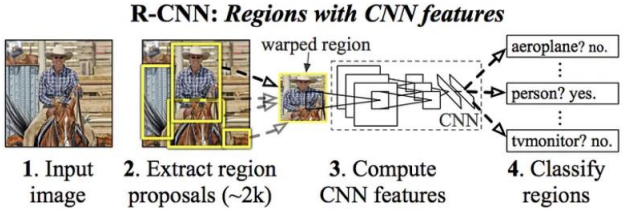

Figure 3. R-CNN Model Architecture

2.3.2 Mask R-CNN

He, Kaiming et al. (2017) defines Mask R-CNNas an extension of Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps. Moreover, Mask R-CNN is easy to generalize to other tasks, e.g., allowing us to estimate human poses in the same framework.

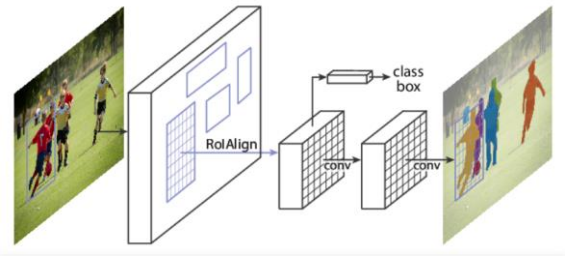

Figure 4. Mask R-CNN Architecture

2.3.3 RetinaNet

Del Prete et al. (2020) states that the one-stage RetinaNet network architecture uses a Feature Pyramid Network (FPN) backbone on top of a feedforward ResNet architecture to generate a rich, multiscale convolutional feature pyramid. To this backbone, RetinaNet attaches two subnetworks, one for classifying anchor boxes and one for regressing from anchor boxes to ground-truth object boxes. The object classification subnet and the box regression subnet share a common structure but use separate parameters.

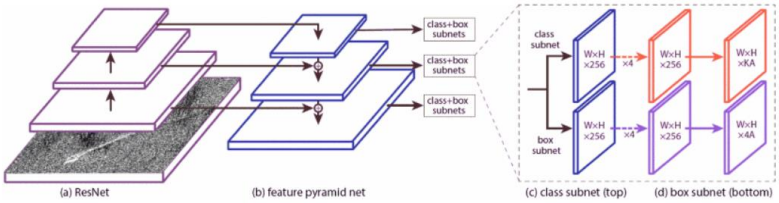

Figure 5. RetinaNet Architecture

3 Model Architecture, Training

3.1 SSD: Single Shot Detector for Object Detection

As stated by Khandelwal (2019), SSD takes only one shot to detect multiple objects present in an image using multibox. It is considered to be significantly faster in speed and better in accuracy in comparison to other object detection algorithms such as the Faster R-CNN model and the YOLO model. The high detection accuracy is attributed to the usage of multiple filters with different sizes, and aspect ratio for object detection. These filters are applied to multiple feature maps from the later stages of a networks which helps perform detection at multiple scales.

Therefore, the SSD architecture was chosen for its ability to perform real-time object detection while maintaining high accuracy.

3.2 Architecture of Single Shot Detector

3.2.1 Base neural network: Extracts features

SSD possesses a base VGG-16 network followed by multibox convolution layers. VGG-16 which is used for feature extraction is a standard Convolutional neural network (CNN) for high quality image classification but without the final classification layers.

3.2.2 Additional convolutional layers: Detect objects

As mentioned previously, additional convolutional layers are added for detection. The size of the convolutional layers at the end of the base network decreases progressively. This aids in the detection of objects at multiple scales. It is important to note that the convolutional model for detection is different for each feature layer.

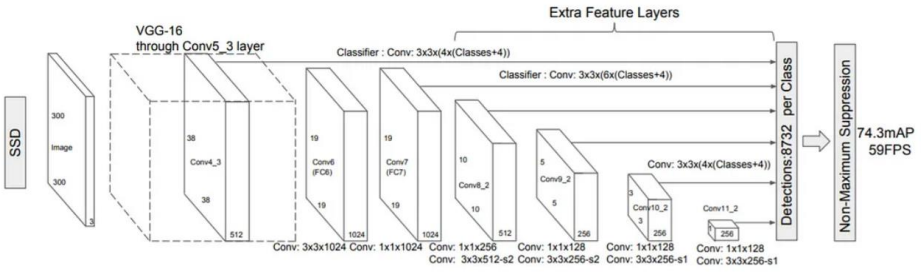

Figure 6. SSD model structure

Khandelwal (2019) clearly states that the prediction for the bounding boxes and confidence for different objects in the image is done by multiple feature maps of different sizes that represent multiple scales. The progressive decrease of convolutional layers reduces the size of the feature map size and increases the depth. The deep layers cover larger receptive fields which in turn construct more abstract representations. As a result, detecting larger objects is made easier. Similarly, the initial convolutional layers cover smaller receptive fields and are helpful in detecting smaller objects in the image.

3.3 Single Shot Detector Model training

3.3.1 SSD Model pre-training

The pre-trained model used for this project is a version of an SSD Model called "ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8". It was pre-trained on the "Common Objects in Context" (COCO) dataset, which is a dataset of 123,287 images containing 886,284 instances of 80 classes. The 80 classes contain the classes of interest for this project: pedestrians, busses, cars, cyclists, and construction sites. Moreover, the model was built with the MobileNetV2 architecture, which is a convolutional neural network architecture for efficient mobile vision applications developed by Google researchers in 2018 that allow to work with limited computational resources. Finally, the model uses Feature Pyramid Network Lite, which is a feature extraction technique allowing the model to detect objects of varying scales and sizes.

3.3.2 Data pre-processing

To fine-tune the afore mentioned pre-trained model for object detection on the 'Annotated Images of Traffic Cameras' dataset, the dataset needed to be processed. The input images are color and therefore need to be converted to tensors with 3 channels of shape based on width and height. The annotations for the objects, containing the location of each object along with class label and other image attributes are stored in XML files. These XML files were converted to the Pascal VOC format, this format requires an XML file for each image in the dataset, containing information about the objects present in that image, including their class, bounding box coordinates, and other metadata. This allowed for training and test data frames to be created based on image id, containing the fields: Image_id, Object_class, x_min, x_max, y_min, y_max. The x and y are the coordinates of the objects, and an image may contain many objects. The last step was to translate both image and label to tensor representation for input to the SSD model.

3.3.3 SSD Model fine-tuning

Once the data was prepared, the model architecture was modified for compatibility, so the training process could begin. The model input architecture was modified to accept the two image sizes stored in the fine-tuning dataset. This was to ensure the model accepted the various images sizes, which were different from the 320x320 pixel dimensions used for training the original model architecture. For the larger images in the fine-tuning dataset, a fixed shape resizer with a height of 288 pixels and a width of 352 pixels was used. Rendering those images consistent with the others in the set.

Model fine-tuning parameters were configured as follows. The activation function of the model was the Rectified Linear Unit (ReLU) with a maximum output of 6. The box predictor used was a weight-shared convolutional box predictor with a depth of 128, 4 layers before the predictor, and a kernel size of 3. The class prediction bias was set to -4.6 and the score threshold used for non-maximum suppression was 9.99e-0.9. The Intersection over Union (IoU) threshold for non-maximum suppression was set to 0.6.

The loss function used was a combination of weighted smooth L1 localization loss and weighted sigmoid focal classification loss with a gamma of 2.0 and alpha of 0.25. The classification and localization weights were set to 1.0. The optimizer used was a momentum optimizer with a cosine decay learning rate of 0.079 and a momentum optimizer value of 0.899. The learning rate was warmed up for the first 1,000 steps with a learning rate of 0.025.

A baseline model was first created to test results found to the out-of-box SSD model, trained on the COCO dataset. This model is not able to detect objects pertaining to 5 possible classes given the pre-processed fine-tuning data as input. To make predictions and generate results, the model architecture was modified as stated above and only 1 training step was performed. This produced a "baseline pre-trained" model that could be used for comparison. To compare various fine-tuned models to the "baseline pre-trained" model, 4 additional versions were created and trained to various degrees.

1. Baseline Model : There was no fine tuning done on this model.
2. V1 : This model used all fine-tuning images, a batch size of 16, and was trained to 50k steps.
3. V2.1 : The model used only images of dimension 352x288, a batch size of 32, and was trained to 200k steps.
4. V2.2 : The model used only images of dimension 352x288, a batch size of 32, and was trained to 25k steps.
5. V2.3 : The model used only images of dimension 352x288, a batch size of 64, and was checkpointed at varying steps, with performance measures taken at each set of checkpoints.

The training procedure for the above model variations came by way of an iterative process. The first model, version 1, yielded substandard results, and intuition led to the decision to remove the images that would need to be passed through the fixed-size re-shaper, presuming that these images were negatively impacting model generalization. Version 2 used only the 352x288 dimension images, and also leveraged a larger batch size, the first instantiation of this model was trained over several days, and results showed that the model began to overfit the training data. To determine the overfitting point, another model was trained quickly, to 25k steps for comparison, which allowed to gauge the ideal training somewhere between these two step ranges (25k and 200k). Model 2.3 utilized model checkpoints to be trained to a high number of steps while saving model states at various training points (26k, 43k, 62k, through to 119k steps). This allowed for a large amount of training information to be stored and evaluated, as well as the ability to rollback a model to a previous state and perform all evaluations on said previous state. At this point, various model metrics were obtained for analysis.

4 Performance Measures

To classify objects in images, an algorithm is used to automatically identify and locate the objects within an image. However, just as humans, these algorithms are inclined to make mistakes and are not perfect. It may detect an object that is not in the picture or they may fail to detect an object that is present in the picture. To evaluate the performance of an object detection algorithm, evaluation criteria's (performance measures) that will quantify its accuracy must be assessed. While there are several performance measures that are commonly used in object detection, the four measures that were used in the case of assessing the algorithms for this project were as follows:

1. Recall
   Recall measures the fraction of the objects that were correctly detected out of all the objects that were present in the image
2. Precision
   Precision measures the fraction of the correctly detected objects out of all the detections made by the algorithm.

By using these performance measures, an informed assessment of the models can be made whilst comparing them to each other.

5 Results

In this section, the results a thorough comparison is made of the performance based on 4 different areas corresponding to the size of detected objects; small, medium, large and all (which is all the objects together independently of their sizes). Moreover, the results were kept only for detections where the IoU is between .5 and .95, which means that we've drawn a bounding box over 50% or more of the ground truth object.

Table 2: Final model performance on training set

Model Formulation IoU Threshold = 0.5 – 0.95 Max Detections = 100				Training Loss Criteria				
Model Code	**Image size**	**Steps Trained**	**Batch Size**	**Class Loss**	**Local Loss**	**Reg. Loss**	**Total Loss**	**Learning Rate**
Baseline	-	-	-	-	-	-	-	-
V1	All images	50k	16	-	-	-	-	-
V2.1	352x288 images	200k	32	-	-	-	-	-
V2.2	352x288 images	25k	32	0.0678	0.0537	0.0833	0.2048	0.0000
V2.3	352x288 to 150k	@26k	64	0.0837	0.0637	0.0590	0.2063	0.0746
V2.3	352x288 to 150k	@43k	64	0.0696	0.0463	0.0427	0.1587	0.0651
V2.3	352x288 to 150k	@62k	64	0.0644	0.0442	0.0341	0.1428	0.0504
V2.3	352x288 to 150k	@89k	64	0.0534	0.0384	0.0273	0.1190	0.0280
V2.3	352x288 to 150k	@101k	64	0.0523	0.0346	0.0247	0.1116	0.0195
V2.3	352x288 to 150k	@109k	64	0.0531	0.0385	0.0233	0.1150	0.0140
V2.3	352x288 to 150k	@119k	64	0.0457	0.0278	0.0220	0.0954	0.0082

Table 3: Final model performance on validation set.

Model Formulation IoU Threshold = 0.5 – 0.95 Max Detections = 100				Validation Loss Criteria			
Model Code	**Image size**	**Steps Trained**	**Batch Size**	**Class Loss**	**Local Loss**	**Reg. Loss**	**Total Loss**
Baseline	-	-	-	-	-	-	-
V1	All images	50k	16	-	-	-	-
V2.1	352x288 images	200k	32	-	-	-	-
V2.2	352x288 images	25k	32	0.1911	0.2586	0.0833	0.5330
V2.3	352x288 to 150k	@26k	64	0.2358	0.1999	0.0590	0.4947
V2.3	352x288 to 150k	@43k	64	0.2726	0.1987	0.0430	0.5142
V2.3	352x288 to 150k	@62k	64	0.2871	0.2036	0.0341	0.5248
V2.3	352x288 to 150k	@89k	64	0.3002	0.1929	0.0275	0.5206
V2.3	352x288 to 150k	@101k	64	0.3037	0.1887	0.0247	0.5172
V2.3	352x288 to 150k	@109k	64	0.3205	0.1985	0.0233	0.5424
V2.3	352x288 to 150k	@119k	64	0.3388	0.1936	0.0220	0.5544

Table 4: Final model performance (average precision) on validation set.

Model Formulation IoU Threshold = 0.5 – 0.95 Max Detections = 100				Validation Average precision			
Model Code	**Image size**	**Steps Trained**	**Batch Size**	**area=small**	**area=medium**	**area=large**	**area=all**
Baseline	-	-	-	-	-	-	-
V1	All images	50k	16	-	-	-	-
V2.1	352x288 images	200k	32	-	-	-	-
V2.2	352x288 images	25k	32	0.1720	0.5080	0.6640	**0.2290**
V2.3	352x288 to 150k	@26k	64	0.1609	0.4639	0.6474	0.2210
V2.3	352x288 to 150k	@43k	64	0.1313	0.4457	0.6738	0.2067
V2.3	352x288 to 150k	@62k	64	0.1544	0.4560	0.6492	0.2122
V2.3	352x288 to 150k	@89k	64	0.1747	0.4434	0.6614	0.2229
V2.3	352x288 to 150k	@101k	64	0.1603	0.4949	0.6359	0.2259
V2.3	352x288 to 150k	@109k	64	0.1825	0.4824	0.6121	0.2298
V2.3	352x288 to 150k	@119k	64	0.1794	0.5071	0.6296	0.2356

Table 5: Final model performance (average precision) on validation set.

Model Formulation IoU Threshold = 0.5 – 0.95 Max Detections = 100				Validation Average recall			
Model Code	**Image size**	**Steps Trained**	**Batch Size**	**area=small**	**area=medium**	**area=large**	**area=all**
Baseline	-	-	-	-	-	-	-
V1	All images	50k	16	-	-	-	-
V2.1	352x288 images	200k	32	-	-	-	-
V2.2	352x288 images	25k	32	0.3300	0.5670	0.6830	**0.3760**
V2.3	352x288 to 150k	@26k	64	0.3120	0.5432	0.6958	0.3633
V2.3	352x288 to 150k	@43k	64	0.3096	0.5804	**0.7250**	0.3662
V2.3	352x288 to 150k	@62k	64	0.3273	0.5418	0.6625	0.3705
V2.3	352x288 to 150k	@89k	64	0.3264	0.5264	0.6917	0.3696
V2.3	352x288 to 150k	@101k	64	0.3219	0.5828	0.6792	0.3700
V2.3	352x288 to 150k	@109k	64	0.3251	0.5375	0.6750	0.3687
V2.3	352x288 to 150k	@119k	64	0.3221	0.5692	0.6708	0.3722

On the validation set, it is evident that the model is performing better for medium and large objects in comparison to small ones. The best model has an average precision of 0.6640 and 0.5080 for large and medium objects respectively compared to 0.172 for small objects. This phenomenon was observed for the recall, with a recall of 0.6830 and 0.5670 for large and medium objects respectively compared to 0.33 for small objects. Even though it is not best model overall on validation set, it obtained a recall of 0.7250 for large object on the validation set with model V.2.3 with @43k steps trained, which is the best recall on all the recalls computed.

This difference can be well understood as the algorithm struggles to detect and recognize small objects that are most of the time less clear that medium on large objects on the images.

6 Conclusion

This project highlights the complexities of working with image data from real-life datasets. Despite using a pre-trained model, the computational power required to produce accurate results was significant. Although the model did not perform as well as expected, it was still useful in identifying objects with a reasonable degree of accuracy.

Moving forward, models could be trained on a larger dataset of Montreal surveillance cameras' images, focusing initially on medium to large objects to extract valuable information. Once the model is improved it can then be refined to focus on smaller objects. This would allow for more precise object detection in the future, which can be beneficial for applications such as traffic monitoring, surveillance, and autonomous driving. An additional potential area for future work could be to employ data augmentation techniques such as rotating, flipping and scaling images. This increases the size of the training dataset and potentially improve the performance of the model.

7 References

[1] Grondin, J.-S.G. (2021) *Detecting objects in urban scenes using yolov5*, Medium. Towards Data Science. Available at: https://towardsdatascience.com/detecting-objects-in-urban-scenes-using-yolov5-568bd0a63c7 (Accessed: April 7, 2023).

[2] Khandelwal, R. (2019) *SSD: Single shot detector for object detection using MultiBox*, Medium. Towards Data Science. Available at: https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca (Accessed: April 7, 2023).

[3] D. Snegireva and A. Perkova, "Traffic Sign Recognition Application Using Yolov5 Architecture," *2021 International Russian Automation Conference (RusAutoCon)*, Sochi, Russian Federation, 2021, pp. 1002-1007, doi: 10.1109/RusAutoCon52004.2021.9537355.

[4] Das, A.K. et al. (2020) "Deep Learning Techniques—R-CNN to Mask R-CNN: A Survey," in *Computational Intelligence in Pattern Recognition Proceedings of CIPR 2019*. Singapore: Springer Singapore, pp. 657–668.

[5] R. Del Prete, M. D. Graziano and A. Renga, "RetinaNet: A deep learning architecture to achieve a robust wake detector in SAR images," 2021 IEEE *6th International Forum on Research and Technology for Society and Industry (RTSI)*, Naples, Italy, 2021, pp. 171-176, doi: 10.1109/RTSI50628.2021.9597297 / (Accessed: April 20, 2023).

[6] Zhang, Y. et al. (2020) *Mask-refined R-CNN: A network for refining object details in instance segmentation*, *Sensors (Basel, Switzerland)*. U.S. National Library of Medicine. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7070892/ (Accessed: April 20, 2023).

[7] He, Kaiming et al. "Mask R-CNN." *2017 IEEE International Conference on Computer Vision (ICCV)* (2017): 2980-2988.