Embedded port infrastructure inspection using artificial intelligence

Embedded Port Infrastructure Inspection using Artificial Intelligence

Nicolas Vigne
THALES Research & Technology
Palaiseau, France
nicolas.vigne@thalesgroup.com

Rémi Barrère
THALES Research & Technology
Palaiseau, France
remi.barrere@thalesgroup.com

Benjamin Blanck
SEC 11 – Research and Development
Hamburg Port Authority
Hamburg, Germany
Benjamin.Blanck@hpa.hamburg.de

Florian Steffens
Technical Department for building inspection and building materials advice
Hamburg Port Authority
Hamburg, Germany
Florian.Steffens@hpa.hamburg.de

Ching Nok Au
Center für Maritime Logistik und Dienstleistungen
Fraunhofer Gesellschaft
Hamburg, Germany
ching.nok.au@cml.fraunhofer.de

James Riordan
School of Computing, Engineering and Physics Sciences
University of the West of Scotland
Glasgow, Scotland
James.Riordan@uws.ac.uk

Gerard Dooly
Department of Electronics & Computer Engineering
University of Limerick
Limerick, Ireland
gerard.dooly@ul.ie

***Abstract*—This paper is related to the H2020 RAPID project, focusing on the AI automated monitoring of critical port infrastructure such as concrete structure. An important objective in RAPID was to translate a technical expertise of labelling cracks into a UAV real-time embedded solution based on deep neural networks. The efficiency of a deep learning algorithm is highly dependent on the data used for training, and this paper illustrates the fact that the use of open-source data is not sufficient. An intensive collaboration between neural network and industry experts made it possible to obtain a relevant data set of sufficient size to carry out quality training. This collaborative work also allowed the definition of ground truths, necessary for the validation of the detection system. In this paper, we provide a definition of the useful metrics and objectives for the algorithms in accordance with the complexity of the cracks and their environment, used to identify the best neural network in terms of efficiency, and performance to embed it on a UAV. Our research then focused on the hardware platform that could be used as an onboard computer for the drone, considering Size, Weight and Power (SWaP) constraints. We applied optimization methods to reduce the latency of our models while maintaining high accuracy. These techniques allowed us achieve a state-of-the-art detection rate while complying with the real-time requirements of the overall system, and the need to increase productivity of mission inspections in a port environment through high-speed inferences.**

Keywords — Crack Detection, Embedded, Artificial Intelligence, Deep Learning, Port Infrastructure Inspection.

I. Introduction

Ports are the nerve center of the global transport system linking land, water and sea and are critical infrastructure for maintaining supply chains across all industries. In order to ensure their proper functioning, it is necessary to set up a structural failure warning system, which includes the detection of cracks and the temporal change in their morphology. The aim of such a system is to minimize the disruption and downtime of supply chains, but also to save lives by preventing major disasters such as bridge collapses like those in Genoa, Italy, or more recently in Morbi, India.

The RAPID project proposes to enable real-time crack detection based on Deep Learning and a UAS-based system. RAPID ensures infrastructure monitoring of hard-to-reach assets without risk to human safety, without increasing logistical costs and eliminated disruptions to traffic typically seen in other approaches such as the use of scaffolding or cherry pickers. The RAPID approach maintains the human at the centre of the decision-making process: to ensure maximum safety, if the algorithm has any doubts about the presence of a crack, it is up to an expert to decide and make the final decision. However, human input dependence is kept to a minimum, which significantly increases the productivity of inspection missions, and will be reduced gradually, as each human decision will be incorporated into the learning base to improve the process over time.

Furthermore, the use of automation and digitization allows for the possibility of coupling it with other complementary functionalities such as photogrammetry. The latter makes it possible to reconstruct the detected cracks in 3D and display them in virtual reality - providing the expert with additional information - as well as automatic monitoring of the size of the cracks. In order to detect an increase in their size: a sign of structural deterioration that must be dealt with quickly.

Our work takes place in this complex context, which requires certain optimizations in order to propose a crack detection system with an accuracy superior to the state of the art, while respecting SWaP (Size, Weight and Power) criteria, allowing it to be embedded, on a drone while maintaining a high degree of autonomy.

II. Necessary and Essential First Steps

The solution is focused on the use of deep learning. For efficient training and accurate prediction, the training dataset is a key component of any neural network-based application design. This dataset conditions the effective performance of such an algorithm. It is therefore possible, with the same

European Commission Innovation and Networks Executive Agency.

algorithm, to obtain very good or very bad results, depending only on the quality of the dataset.

A. *Task decision*

Deep neural networks are used across many industry sectors and appear to be well suited to the needs of the project, based on imagery. There are three main tasks in deep learning computer vision: classification, object detection and semantic segmentation [1]. We have already found solutions in the State-of-the-art [2] which achieve high level of accuracy for crack detection and have been able to test each task with open-source crack data [3]. It turned out that real cases applications were very different from the available open-source data, making it almost impossible to detect cracks in an environment such as the port of Hamburg. Figure 1 depicts how real cases images are different from the open-source ones.

Figure 1: open-source crack image (left), actual lock image (right).

It was then necessary to design a best suited dataset, with joint efforts from the Hamburg Port Authority (HPA). Our objective was to replicate professional knowledge, so HPA’s expertise was a key component for data creation.

After considering the technical needs of the other stakeholders in the RAPID project, we decided that object detection was the proper task to use. Actually, classification would have required highly zoomed images to be selective enough. Semantic segmentation, on its side, was not a realistic solution. Labelling semantic masks would have been too expensive in terms of computing power. In addition, segmentation provides more information than what was required for the project. As we only wanted to know the locations of the cracks, we did not need the exact shape of the cracks.

B. *Dataset acquisition and collaboration with Hamburg Port Authority*

HPA acquired thousands of images of the surface of a quay wall with a high-resolution camera in March 2021. We selected only a part of these images to be used for our dataset, with the objective to reduce overlap as much as possible and avoid overfitting of the model. Later on, we managed to acquire more data, which adds up to a dataset of 726 images with 4311 annotated instances of cracks.

In order to be able to perform an efficient training for the neural networks, at least 1500 instances of cracks are necessary. Moreover, at least 500 images per class are required to avoid overfitting. In our case, we succeeded to fulfil these needs.

C. *Labelling tools and annotation standardisation*

When dealing with an object detection problem, the labels are called bounding boxes. Their standard format is defined by the COCO guidelines [4].

The labelling of the 726 images was a collaborative work between THALES and HPA. Having the expertise from the algorithmic point of view, THALES had a better idea of what was needed as labels for a neural network. On the other side, HPA being the expert in terms of structural cracks, their experience was necessary to identify the cracks on the wall. We used the open-source tool labelImg [5] to produce the required bounding boxes.

A first training with these data showed that the outputs were incorrect and presented a lot of overlap between the predicted boxes, because the first annotations had bounding boxes that were too small. This point illustrates the need for high quality training data. We worked to adapt the annotations in a collaborative way, thus making them optimized for the neural network training.

III. ALGORITHM SELECTION

Once the annotated dataset was completed and data structure in place, then we moved towards model selection. We identified several algorithms for use which were suited to the RAPID requirements. Using the candidate models and prepared dataset, we trained a number of smart algorithms and tested them with specific metrics functions. Then we applied optimization methods to reduce the inference time, making it possible to embed the algorithms on a low-power hardware platform.

A. *Candidate Algorithms*

As we focused on object detection, we decided to benchmark the most widely used models for embedded detection You Only Look Once (YOLO) algorithms, namely: YOLOv4 [6], YOLOv5, YOLOv6 [7] and YOLOv7 [8].

The architecture of these models is on the same structure: they are composed of a backbone, a neck and a head.

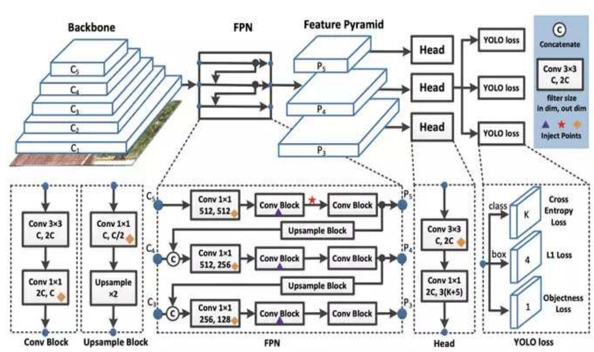

Figure 2: YOLOv4 architecture.

The backbone is the “body” of the network, which will allow all the decisions taken by it. To simplify, it can be seen as a “converter” that converts the input image, an as-is data format difficult to process by AI, into a set of information that characterizes its content called “features” (such as the presence of shapes, colours, textures, …) from which it is easy to recognize objects. It is thus composed of a series of successive layers, and extracts feature maps, i.e. maps indicating which features are present at which locations in the image. The role of the "neck" is to extract the relevant features from all the layers of the backbone, and to combine them into features useful for our detection task. Finally, the "head" is responsible for the final decision of the network. Based on the information provided by the neck, it decides where the elements to be detected are, by drawing bounding boxes around them, and what they are, by giving the nature of the object present in each bounding box.

The difference between the various versions of the YOLO algorithm can be subtle. YOLOv4 was the state-of-the-art of object detection back in 2020 at the beginning of the project. YOLOv5 is an improved version of YOLOv4. Indeed, these models rely on anchor boxes to predict the bounding boxes size, the ratio of these anchor boxes is defined prior to the training step. YOLOv5 is provided with a genetic algorithm to compute the best anchor boxes ratio to fit the training dataset. Therefore, YOLOv5 tends to converge to higher accuracies than YOLOv4. YOLOv6 appeared at the beginning of 2022 with a quantization aware training friendly implementation, making it a good choice for real-time embedding problems. Finally, YOLOv7 paper was published in July 2022. It has two major architectural reforms: the integration of an Extended Efficient Layer Aggregation Network and a Model Scaling for Concatenation-based Models.

These algorithms come with different characteristics as reported by Table 1. The difference between all these models is the amount of convolution layers in the head and the backbone, jointly with the input size and kernel size of each layer. The 2D convolution computational complexity is $O(n^4)$. Consequently, increasing the hyper-parameters of each convolution layer results in an increase for parameters and the number of floating point operations (Flops).

We therefore dispose of several algorithms offering a trade-off between accuracy and computing cost that we can evaluate.

Models	COCO mAP (%)	Flops (G)	Params (M)
YOLOv4 tiny	32.2	6.9	6.1
YOLOv4 csp	50.0	120.0	60.0
YOLOv4-P5	51.6	157.2	78.2
YOLOv4-P6	54.0	282.5	140.6
YOLOv5n	28.0	4.2	1.8
YOLOv5s	37.4	15.8	7.0
YOLOv5m	45.4	47.9	20.8
YOLOv5l	49.0	107.8	46.1
YOLOv5x	50.7	204.0	86.2
YOLOv6n	37.5	11.1	4.3
YOLOv6s	45.0	44.2	17.2
YOLOv7	51.4	103.2	36.5
YOLOv7x	53.1	188.0	70.9

Table 1: Benchmarked algorithms.

B. Selection method and criteria

In order to define the ability of an object detection algorithm to detect the cracks properly, we need to evaluate them on several metrics. These metrics relies on the components of the confusion matrix: True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN).

1) Recall

The recall defines the preferred metric so that an algorithm does not miss the detection of a crack.

$$Recall = \frac{TP}{TP+FN} \qquad (1)$$

A recall with a value of one means that the neural network does not emit any false negatives, as shows the equation (2).

$$\lim_{FN \to 0} \frac{TP}{TP+FN} = 1 \qquad (2)$$

This metric is the most important one. Indeed, Thales and HPA agreed to state the fact that it is worse to miss a crack than to detect one when there is none.

2) Precision

The precision defines how well an algorithm's predictions are actual cracks. A precision of one means that an algorithm does not emit false positives.

$$Precision = \frac{TP}{TP+FP} \qquad (3)$$

3) Intersection over Union

Traditionally, this parameter is set to a certain threshold for the whole training of a model. If a prediction has an Intersection over Union (IoU) higher than the mentioned threshold, we consider the prediction effective and we can allocate a class to the predicted box. In object detection, this metric tells if a box is properly placed.

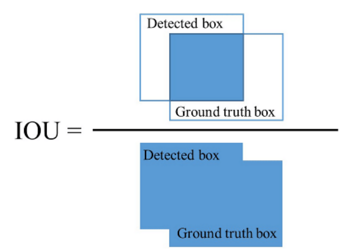

Figure 3: Intersection over Union.

4) mean Average Precision

If we refer to the MS COCO dataset [4] made up of 80 classes, the mAP is the average of the accuracies observed for these 80 classes. In the RAPID case study, there is only have one class; the mAP is therefore equal to the AP of the "crack" class.

We can retain two different types of mAP. The mAP at 0.5 and the mAP at 0.5:0.95. The first one considers a predicted box as effective if the IoU is equal or higher than 0.5 and the second one considers bounding boxes that bear an IoU included between 0.5 and 0.95. The second case helps us to avoid overfitting. Indeed, if a lot of the IoU are over 0.95 the mAP score will be low. These two mAP are the standard usual metrics in detection neural networks.

5) Precision-Recall curve

A precision-recall curve [9] (or PR Curve) is a plot of the precision (y-axis) and the recall (x-axis) for different probability thresholds. A model with perfect skill will be depicted as a point at a coordinate of (1,1), a skilful model will be represented by a curve that bows towards a coordinate of (1,1) and a no-skill classifier will be a horizontal line on the plot with a precision that is proportional to the number of positive examples in the dataset. For a balanced dataset, this will be 0.5. The focus of the PR curve on the minority class makes it an effective diagnostic for imbalanced binary classification models.

C. Training strategy

In this part, we describe all the necessary hyper-parameters used for the training step.

The selected models are batch-size agnostic, indeed what matters is the amount of epochs for the training. This means that if we train the algorithm long enough, it will eventually converge to the same level of precision, regardless of the batch size.

An object detection algorithm needs two losses functions. The IoU loss function is the Complete IoU (CIoU) Loss [10], this loss function uses three geometric factors. Firstly, the overlap area between the predicted box and the ground truth bounding box-IOU loss. Then, the central point between the predicted box and the ground truth bounding box-DIoU loss. Finally, an aspect ratio of the predicted box and the ground truth box. While the class loss function, is the Cross Entropy Loss [11]. Since we used only one class, the class loss function was not relevant.

We trained each model for 300 epochs. The best fit were chosen from the lowest validation loss during the training. In order to shorten the training when it was necessary, we also used an early stopping [12]. It implied that if the model had not improved for 100 epochs, the training stopped.

We used a Stochastic Gradient Descent (SGD) [13] optimizer. When training a model for a large amount of epoch, the SGD optimizer turns out to perform better than the Adam optimizer does [14]. This behaviour was also verified through experiments.

The data augmentation techniques used with these algorithms were the following: hue, saturation and value augmentation, rotation, translation, rescaling, shear, perspective, flip-up/down, flip-left/right, mosaic, mixup and copy paste augmentation. Once again, this augmentation pipeline made our models more robust [15].

Each one of these models were pre-trained on the COCO dataset. This method has been proven to allow the neural network to converge faster rather than training from random value weights [16]. We previously mentioned that a model trained on the open-source crack data does not detect any crack on the real use-case data. Therefore, there is no need to pre-train our algorithms on this specific dataset.

D. Results and selection of best candidates

After each training, it was possible to establish the following results:

Models	mAP (%)	Recall (%)	Flops (G)	Params (M)
YOLOv4 tiny	53.8	51.9	6.9	6.1
YOLOv4 csp	88.8	88.4	120.0	60.0
YOLOv4-P5	91.0	89.4	157.2	78.2
YOLOv4-P6	92.0	90.4	282.5	140.6
YOLOv5n	94.0	90.6	4.2	1.8
YOLOv5s	94.9	94.1	15.8	7.0
YOLOv5m	96.6	96.1	47.9	20.8
YOLOv5l	97.0	96.6	107.8	46.1
YOLOv5x	97.8	95.7	204.0	86.2
YOLOv6n	82.9	80.0	11.1	4.3
YOLOv6s	88.2	87.5	44.2	17.2
YOLOv7	94.5	94.1	103.2	36.5
YOLOv7x	93.9	92.3	188.0	70.9

Table 2: Benchmarked models after training on our data.

We can notice that the YOLOv5 algorithms succeed to converge to high accuracies and interesting recall average values. They have many alternatives and present interesting trade-offs between the accuracy, the training time and the inference time. YOLOv5 outperform YOLOv4 due to the anchor boxes genetic algorithm.

The YOLOv7 algorithms also reach high accuracies. We will keep these two series of models for the embedding part of this project.

Here are the training curves of YOLOv7 as an example:

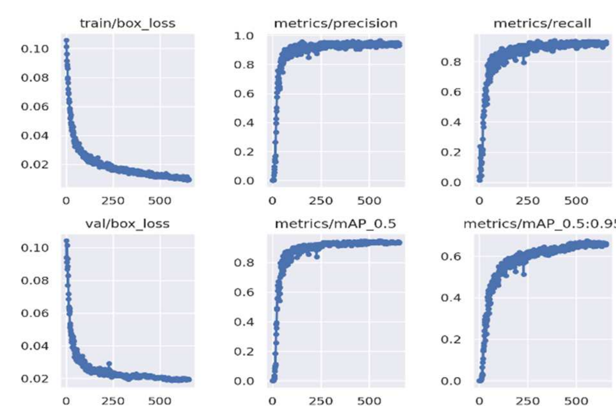

Figure 4: YOLOv7 training curves.

Here, on Figure 5, we can have an example of the PR curve for YOLOv7. As we can observe, the curve tends to the (1,1) coordinate. This means that we can qualify this model as skilful and produce a very little share of false positives and false negatives.

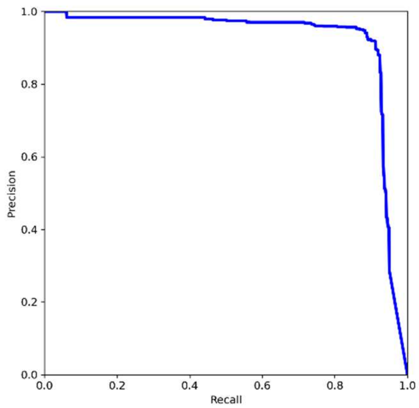

Figure 5: Precision-Recall curve of YOLOv7.

Our results with most of the algorithms reached a higher accuracy than the state-of-the-art [2]. The data remains a decisive point to design a more robust algorithm. As our object detection algorithms has trained on 726 images, they might be sensitive to a more varied environment. It is through iterative training throughout the use of the whole system that these algorithms will tend to be more robust and accurate.

E. *Real-time optimization on hardware platforms*

Now that we have a performing solution, we need to embed it on a hardware target. Through this section, we will see what we can do to reduce the computing load of these models.

1) *Hardware platforms*

For the RAPID project, we explored the NVIDIA hardware platforms suite. Indeed, as the other partners are using the NVIDIA boards, we wanted to satisfy the consortium and propose a technical solution that could be compatible with the other partner's technical solutions. We also tested the Hailo8 chip, which happens to be a promising solution. With an architecture that takes advantage of the core properties of neural networks, this neural chip allows edge devices to run deep learning applications at full scale with efficiency and sustainability while significantly lowering costs.

Table 3 presents the main characteristics of these platforms: trillions of operations per seconds (TOPS), bandwidth and Thermal Design Power (TDP).

Platforms	TOPS	Bandwidth (GB/s)	TDP (W)
Xavier NX	21	59.7	20
Xavier AGX	32	136.5	50
Orin AGX	275	275	60
Hailo8	26	-	8.65

Table 3: Studied platforms specifications.

2) *Optimization methods*

With these selected boards to embed our neural networks, we needed to apply optimization methods to make them more efficient by reducing both the inference latency and the power consumption. We also studied the effect of these methods on the accuracy.

a) Floating point 16 half precision conversion

The default precision of a neural network's weights is floating point 32. Converting the weights of the model to floating point 16 precision have been proven to be an efficient way of reducing the computational cost of an algorithm without a significant deterioration of accuracy [17].

b) Post Training Quantization

The post training quantization (PTQ) [18] consists in converting the fp32 weights of the model in int8 precision. This optimization method happens after the training of the model. Post Training Quantization computes a scale after the network has been trained. A representative dataset is used to capture the distribution of activations for each activation tensor, then this distribution data is used to compute the scale value for each tensor. Each weight's distribution is used to compute the weight scale.

The quantization scheme for the weights is as follow: First, we defined the scale s:

$$s = \frac{\max(abs(x_{min}), abs(x_{max}))}{127} \qquad (4)$$

Where x_{min} and x_{max} were the floating points minimum and maximum values for the weight tensor. Then we quantized each weights value, where x was the floating-point weight value:

$$x_q = roundWithTiesToEven(clip(\frac{x}{s}, -128{,}127)) \qquad (5)$$

c) Quantization Aware Training

Quantization Aware Training (QAT) [18] aims at computing scale factors during training. Once the network is fully trained, Quantize (Q) and Dequantize (DQ) nodes are inserted into the graph following a specific set of rules. The network is then further trained for few epochs in a process called Fine-Tuning. Q/DQ nodes simulate quantization loss and add it to the training loss during fine-tuning, making the network more resilient to quantization. In other words, QAT is able to better preserve accuracy when compared to PTQ [18].

d) Hailo dedicated optimization methods

Along with its chip, Hailo offers a dedicated optimization. We can leverage through two parameters: a compression level and an optimization level.

The compression within the Hailo framework corresponds to a quantization of the model in int8 and int4 precision. The level of compression will dictate what share of the model's weights are quantized in int4 precision. The higher the compression level, the faster the inference will be. However, a high compression level tends to reduce the accuracy. The standard compression level is 60% of the weights quantized in int8 precision, while 40% are quantized in int4 precision.

To counter-effect the decrease in accuracy of the compression, Hailo also offers an optimization level. In the same spirit as the compression, we can use different level of optimization. These levels make use of several optimization techniques: Equalization [19], Iterative bias correction (IBC)

[20], Fine-tuning with 4 epochs, Adaround [21] with 320 epochs on int4 layers (level 3) and finally, Adaround with 320 epochs on all layers (level 4).

	Level 0	Level 1	Level 2	Level 3	Level 4
Equalization	✓	✓	✓	✓	✓
IBC		✓			
Fine-tuning			✓		
Adaround				✓	✓

Table 4: Optimization levels of the Hailo framework.

e) Results

From an implementation point of view, the model deployed on NVIDIA platforms and the Hailo chip needed to be implemented respectively with the TensorRT and the HailoRT framework. Then, the next step of our study was to apply every optimization method to the embedded algorithms, and measure their accuracies, latency and power consumption on each hardware platform.

We can monitor the power modes of every NVIDIA platform. A power mode defines the maximum power consumption and the amount of cpu cores used by the device. When it comes to high-consuming platforms, a low power mode will limit the device's performances. In our use-case, we wanted to compare them under the same conditions. Therefore, all these devices were set to a common power mode: 15W – 4 cores.

Table 4 shows the results of the benchmark of each embedded model on the Jetson NVIDIA Xavier NX. As we can see, this board has a low power consumption. The inference speed remains sufficiently high to assess real-time challenges. Then, we have lead the same benchmark on the NVIDIA Xavier AGX board (Table 5). As this is a more powerful board than the Xavier NX is, the inference latencies are reduced. However, the power consumption is also significantly higher. This will be a concern in a production context. Finally, to conclude the benchmarks on the NVIDIA platforms, Table 6 shows the SWaP constraints on the Orin AGX set to 15W and four cores. Even though the Orin AGX is the most powerful platform of this benchmark, the inferences are not as fast as they are on the Xavier AGX. The chosen power mode is too restrictive for such a device. In fact, the Orin AGX needs more power to be exploited at its full potential, as the GPU is very limited in this mode and preponderant in terms of computing time.

Models	fp32			fp16			int8		
	mAP (%)	Latency (ms)	Power (W)	mAP (%)	Latency (ms)	Power (W)	mAP (%)	Latency (ms)	Power (W)
YOLOv5n	94.0	35.7	4.6	93.6	17.8	4.4	92.3	14.1	4.0
YOLOv5s	94.9	37.1	5.6	94.4	21.9	4.6	92.1	14.7	4.1
YOLOv5m	96.6	73.0	6.8	96.3	24.3	5.3	94.6	15.1	4.8
YOLOv5l	97.0	133.9	7.7	97.3	39.9	5.7	94.8	23.2	5.1
YOLOv5x	97.8	240.3	8.8	97.7	71.7	6.5	92.4	38.9	5.9
YOLOv7	94.5	128.2	7.4	95.0	38.2	5.5	86.4	22.2	4.9
YOLOv7x	93.9	221.5	8.1	93.8	66.1	6.0	39.5	35.9	5.4

Table 5: Results of deployment on NVIDIA Xavier NX.

Models	fp32			fp16			int8		
	mAP (%)	Latency (ms)	Power (W)	mAP (%)	Latency (ms)	Power (W)	mAP (%)	Latency (ms)	Power (W)
YOLOv5n	94.0	47.3	5.5	91.6	7.0	11.3	90.3	5.5	10.2
YOLOv5s	94.9	42.5	6.2	91.9	11.7	12.9	89.7	7.8	11.6
YOLOv5m	96.6	85.6	6.9	95.3	27.2	13.6	93.6	16.9	12.2
YOLOv5l	97.0	153.5	7.2	95.6	44.4	13.9	92.9	25.8	12.5
YOLOv5x	97.8	283.3	8.3	95.6	87.5	14.4	90.8	47.5	13.0
YOLOv7	94.5	91.3	6.5	92.3	44.2	14.1	83.9	26.9	12.9
YOLOv7x	93.9	135.7	7.0	91.7	77.6	13.8	38.6	44.7	12.6

Table 6: Results of deployment on NVIDIA Xavier AGX.

Models	fp32			fp16			int8		
	mAP (%)	Latency (ms)	Power (W)	mAP (%)	Latency (ms)	Power (W)	mAP (%)	Latency (ms)	Power (W)
YOLOv5n	94.0	26.2	8.9	91.8	7.9	11.4	91.0	7.2	10.7
YOLOv5s	94.9	40.3	9.3	91.9	13.7	12.1	91.2	9.4	11.1
YOLOv5m	96.6	83.2	9.5	95.3	32.7	12.5	93.8	20.3	11.6
YOLOv5l	97.0	132.0	10.1	95.6	52.8	13.2	93.4	29.9	11.7
YOLOv5x	97.8	249.9	10.8	95.8	90.7	13.0	91.3	53.2	12.2
YOLOv7	94.5	116.0	9.6	92.3	46.3	13.2	90.5	31.6	11.9
YOLOv7x	93.9	179.5	10.1	91.9	83.6	13.1	55.3	50.2	11.9

Table 7: Results of deployment on NVIDIA Orin AGX.

We also tested a few YOLOv5 algorithms on the Hailo8 chip (Table 7). The results shows that the power consumption is very low, allowing this chip to be easily embeddable, without imposing too long inference times. In fact, 37.4 milliseconds remains a real-time application. Finally, the optimization steps suggested by the Hailo documentation offer very few decreasing in accuracies.

Models	RTX3090		Hailo8		
	fp32		Optimization level 2, Compression of 60% int8 and 40% int4		
	mAP (%)	Latency (ms)	mAP (%)	Latency (ms)	Power (W)
YOLOv5s	94.9	7.3	93.8	10.9	1.7
YOLOv5m	96.6	9.8	94.5	19.1	2.1
YOLOv5l	97.0	11.7	95.0	37.4	2.2

Table 8: Results of deployment on the Hailo8 chip.

F. *Expert in the loop: continuous improvement*

The final point of the design of the embedded neural network is that it needs to be trained regularly, since the cracks in the dataset change through time. They do not look the same considering the weather, the lightning, the water level and they may structurally change over time. We can state that inferring a known crack several years after the initial training is the same as inferring an unknown image.

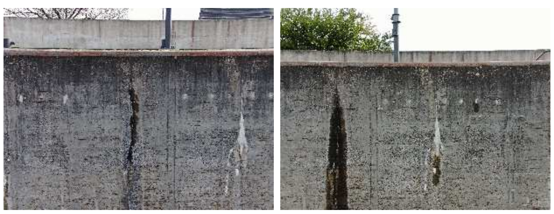

Figure 6: Comparison of the same area taken the 03/17/2021 (left) and the 09/30/2022 (right).

In order to continuously improve the algorithm and reduce the False Negative rate while maintaining maximum efficiency, we save the inferred image for an offline inspection every time the confidence level of a prediction is under a certain threshold. We also save the inferred image if no cracks are detected at all. Thus, a technical expert can process the saved images to make sure the predictions are correct, but limited to those cases that will be reduced as the database grows. Once these annotations are corrected, a script can be called to re-train the model with these new data - if necessary.

IV. Drone integration

Once the neural networks are embedded on a hardware target, the last step is to deploy the technical solution on the UAV. The considered onboard computer is the NVIDIA Jetson Xavier NX, selected through SWAP analysis. In fact, it has a lower power consumption than the other NVIDIA platforms and fits the consortium needs.

To do so, we used the standard tool to integrate technical solutions to a robot: Robot Operating System (ROS). DJI also developed an onboard software development kit (OSDK) in order to easily integrate to the drone and use the various available communication channels. The OSDK provides two tools in order to communicate with the drone: the rosservices, which allows us to control the drone, and the rostopics, which allows us to get and provide information to the drone.

Our goal is to capture three images of a given area of the quay wall with the drone's payload, then use the neural network to predict the crack locations and feed them to the ROS system along with the GPS position and gimbal angle. This set of data and the live video stream are subsequently read by another ROS node and relayed to the consortium's web-based traffic management system, the Service Layer [22], whenever internet connection is available. In this way, other partners in the RAPID project can keep track the exact location of the cracks and even adapt the mission on the fly to include a more thorough scan in a particular area of interest. Figure 7, here under, shows a sample of the drone inspection for crack detection: the drone reaches points 1, 2, 3 and 4 and makes inferences of the respective areas.

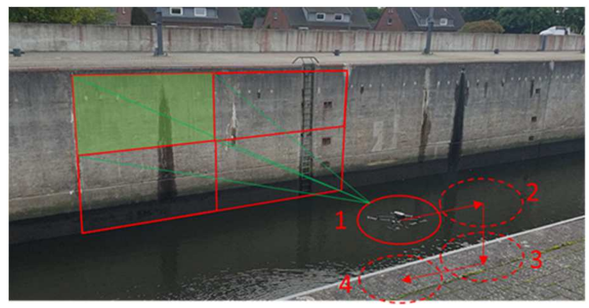

Figure 7: Drone inspection for crack detection.

Thanks to the low latency of our onboard solution, the time taken for the drone to move through its points of passage completely masks the inference time: the detection of cracks is therefore done in real time without impacting on the duration of the inspection mission.

V. Conclusion

Through this paper, we focused on embedded crack detection via deep neural networks and produced a methodology that can translate a technical expertise into an automated solution. The most important point was to reach a very low False Negative rate; this goal is reached through a high Recall. We succeed to obtain higher accuracies than the state-of-the-art [2] while using fewer resources from a data aspect. Finally, due to its energy efficiency, we have succeeded in embedding this system in an environment with large SWaP constraints (UAV), while allowing real-time inference.

We thus propose a complete framework for embedded crack detection, from data acquisition to UAV deployment. As a solution of the RAPID project, the YOLOv5x algorithm in fp16 precision with the Xavier NX is the best candidate, as it offers the same detection rate as the full accuracy model and its inference time is low enough for our application while consuming very little energy.

Figure 8 represents an example of the system output with our embedded neural network.

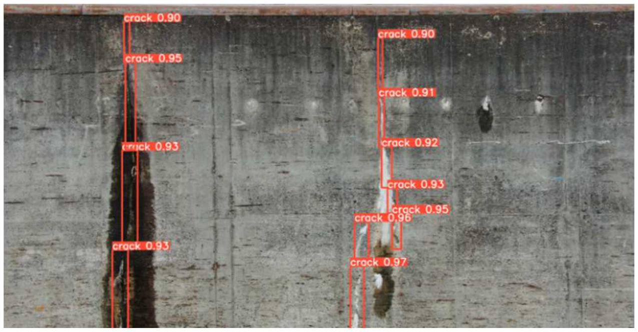

Figure 8: Inferred image of the lock wall.

The future prospects of this study is to aim for more efficient platforms than the NVIDIA solution, so that this solution can move beyond the POC stage and be ported to a production context. The Hailo8 chip paves the way for such a roadmap.

REFERENCES

[1] Viitaniemi Ville, Laaksonen Jorma. (2008). *Techniques for Image Classification, Object Detection and Object Segmentation*. Visual Information Systems. Web-Based Visual Information Search and Management. VISUAL 2008. Lecture Notes in Computer Science, vol. 5188.

[2] Yokoyama Suguru, Matsumoto Takashi. (2017). *Development of an Automatic Detector of Cracks in Concrete Using Machine Learning*. Procedia Engineering, vol. 171, pp 1250-1255.

[3] Sattar Dorafshan, Robert J. Thomas, Marc Maguire. (2018). *SDNET2018: A concrete crack image dataset for machine learning applications. Utah State University*. Data in Brief, vol. 21, pp 1664-1668.

[4] Lin TY, et al. (2014). *Microsoft COCO: Common Objects in Context*. Computer Vision – ECCV 2014. Lecture Notes in Computer Science, vol. 8693.

[5] Tzutalin, *LabelImg,* Git code https://github.com/tzutalin/labelImg, (2015).

[6] Bochkovskiy Alexey, Wang Chien-Yao, Liao Hong-yuan. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. Computing Research Repository, vol. abs/2004.10934.

[7] Li Chuyi, Li Lulu, Jiang Hongliang, et al. (2022). *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. Computing Research Repository, vol. abs/2209.02976.

[8] Wang Chien-Yao, Bochkovskiy Alexey, Liao Hong-Yuan. (2022). *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.* Computing Research Repository, vol. abs/2207.02696.

[9] Davis Jesse, Goadrich Mark. (2006). *The Relationship Between Precision-Recall and ROC Curves*. Proceedings of the 23rd International Conference on Machine Learning, ACM. 06.

[10] Zheng Zhaohui, Wang Ping, Ren Dongwei, Liu Wei, Qinghua Hu, et al. (2021). *Enhancing Geometric Factors in Model Learning and Inference for Object Detection and Instance Segmentation*. IEEE Transactions on Cybernetics, vol. 52, no. 8, pp. 8574-8586 .

[11] Zhilu Zhang, Mert R. Sabuncu. (2018). *Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels.* Computing Research Repository, vol. abs/1805.07836.

[12] Bai Yingbin, Yang Erkun, Han Bo, Yang Yanhua, et al. (2021). *Understanding and Improving Early Stopping for Learning with Noisy Labels*. Computing Research Repository, vol. abs/2106.15853.

[13] Sebastian Ruder. (2016). *An overview of gradient descent optimization algorithms*. Computing Research Repository, vol. abs/1609.04747.

[14] Zhou Pan, Feng Jiashi, Ma Chao, Xiong Caiming, Hoi Steven Chu Hong, E Weinan. (2020). *Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning*. Advances in Neural Information Processing Systems, vol. 33, pp 21285-21296.

[15] Mingle Xu, Sook Yoon, Alvaro Fuentes, Dong Sun Park. (2023). *Image Data Augmentation for Deep Learning: A Survey*. Pattern Recognition, vol. 137.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (2015). *Deep Residual Learning for Image Recognition*. 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778.

[17] Micikevicius Paulius, Narang Sharan, Alben Jonah, Diamos Gregory, Elsen Erich, Garcia, et al. (2017). *Mixed Precision Training*. Computing Research Repository, vol. abs/1710.03740.

[18] Wu Hao, Judd Patrick, Zhang Xiaojie, Isaev Mikhail, Micikevicius Paulius. (2020). *Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation*. Computing Research Repository, vol. abs/2004.09602.

[19] Meller Eldad, Alexander Finkelstein, Uri Almog, Mark Grobman. (2019). *Same, Same But Different - Recovering Neural Network Quantization Error Through Weight Factorization*. Proceedings of the 36th International Conference on Machine Learning, vol. 97, pp 4486-4495.

[20] Finkelstein Alexander, Almog Uri, Grobman Mark. (2019). *Fighting Quantization Bias With Bias*. Computing Research Repository, vol. abs/1906.03193.

[21] Nagel Markus, Amjad Rana Ali, Van Baalen Mart, Louizos Christos, Blankevoort Tijmen. (2020). *Up or Down? Adaptive Rounding for Post-Training Quantization*. Proceedings of the 37th International Conference on Machine Learning, vol. 119, pp 7197-7206.

[22] Au Chin Nok, Delea Cosmin, Schneider Vincent Emmanuel, Oeffner Johannes, Jahn Carlos. (2021). *A web-oriented architecture for deploying multiple unmanned vehicles as a service*. TransNav: International Journal on Marine Navigation and Safety of Sea Transportation, 15(1), 155-164.