# Robust Underwater Image Classification using Image Segmentation, CNN, and dynamic ROI Approximation

**Stefan Bosse[1], Parth Kasundra[2]**
[1]University of Bremen, Dept. Mathematics & Computer Science, Bremen, and Institute for Digitization, Bremen, Germany, [2]marinom GmbH, Bremen

*Abstract* — **Finding classified rectangular Regions of Interest (ROI) in underwater images is still a challenge, moreover if the images pose low quality with respect to illumination conditions, sharpness, and noise. These ROIs can help to find relevant regions in the image quickly by humans, or they can be used as an input for automated structural health monitoring (SHM). This task itself should be done automatically, e.g., used for underwater inspection. Underwater inspection of technical structures, e.g., piles of sea mill energy harvester, typically aims to find material changes of the construction, e.g., rust or coverage with pocks, to make decisions about repair and to assess the operational safety. We propose and evaluate a hybrid approach with segmented classification using small-scaled CNN classifiers (with less than 20000 hyper parameters and 3M unity vector operations) and a reconstruction of labelled ROIs by using an iterative mean and expandable bounding box algorithm. The iterative bounding box algorithm combined with bounding box overlap checking suppress spurious wrong segment classifications and represent the best and most accurate matching ROI for a specific classification label, e.g., surfaces with pocks coverage. The overall classification accuracy (true-positive classification) with respect to a single segments is about 70%, but with respect to the iteratively expanded ROI bounding boxes it is about 90%.**

*Keywords — Image Classification; Region-of-interest detection; Underwater;*

## 1. Introduction

The underwater inspection of technical structures, e.g., construction parts of off-shore wind turbines like piles, involves the identification of various parts in the underwater images. In this work using given videos/pictures the following things can be included:

1. Background with water, bubbles, and fishes, summarized as feature class $B$;
2. Technical structure, e.g., a mono pile of a wind turbine, summarized as feature class $P$;
3. Formation of coverage with marine vegetation or organisms on the surface of the structure, summarized as feature class $C$.

Currently, for the inspection of the mono pile, divers have to go under water. But even if humans inspect the underwater surfaces (underwater by the diver or remotely), the scenes are cluttered and the identification of surface coverage is a challenge. Automated visual inspection is desired to reduce maintenance and service times.

Finding classified rectangular Regions of Interest (ROI) in underwater images is still a challenge. These ROIs can help to find relevant regions in the image quickly by humans, or they can be used as an input for automated structural health monitoring (SHM). This task itself should be done automatically, e.g., used for underwater inspection. Underwater inspection of technical structures, e.g., piles of sea mill energy harvester, typically aims to find material changes of the construction, e.g., rust or coverage with pocks, to make decisions about repair and to assess the operational safety.

The images taken from video recording during diving contain typical changing and highly dynamic underwater scenes consisting of ROIs related to the above introduced classes background (not relevant), technical construction surface, and modified surfaces (rust/coverage), with the highest relevance.

The aim is the development of an automatic bounded region classifier that is at least able to distinguish between background, construction, and construction + coverage classes. The challenge is the low and varying image quality that typically appears in North- and East-sea underwater imaging. The images, typically recorded by a human diver or an AUV, pose low contrast, varying illumination conditions and colours, different viewing angles and spatial orientation and scale, overlaid by mud and bubbles (e.g., from the air supply), and optical focus issues.

We propose and evaluate a hybrid approach with segmented classification using small-scaled CNN classifiers (with less than 10 layers and 100000 hyper parameters) and a reconstruction of labelled ROIs by using an iterative mean and expandable bounding box algorithm. The iterative bounding box algorithm combined with bounding box overlap checking suppress spurious wrong segment classifications and represent the best and most accurate matching ROI for a specific classification label, e.g., surfaces with pocks coverage. The overall classification accuracy (true-positive classification) with respect to a single segments is about 70%, but with respect to the iteratively expanded ROI bounding boxes it is about 90%.

The image segment classification and ROI detection algorithms should be capable to be implemented on embedded systems, e.g., directly integrated in camera systems with application specific co-processor support.

The aim is to achieve an accuracy of at least 85-90% for the predicted images, with a high degree of generalization and independence from various image and environmental parameters such as lighting conditions and background colouration, as well as relevant classification features.

## 2. Related Work

Overall, there is a lack of freely available image data sets from the underwater area, discussed by Chongyi Li et al. [3]. A possible image data set from the underwater area can be found in the Underwater Image Enhancement Benchmark (UIEB DS). These are not specifically technical components such as ship hulls. Further data sets for underwater images are, for example, the Fish4Knowlege data set (Fish4Knowledge DS) for the detection and acquisition of underwater targets; underwater images in the SUN data set for scene recognition and object detection (SUN DS); MARIS data set for Autonomous Marine Robotics (MARIS DS); Sea-thru data set with 1100 underwater images with range maps; Haze-line data set with raw images, TIF files, camera calibration files and range maps [4]. Chongyi Li et al. however, criticize the listed data sets that they usually have monotonous content, limited scenes, few degradation features and, above all, lack of reference images. Also Mittal et al. criticize a non-existent large-scale dataset, which is why they use data augmentation, since it is impossible to train CNN with scarce data [2]. Another dataset can be found on ImageNet database, created by Yifeng Xu et al. [5]. The database ImageNet 2012 included 1000 classes with 15 million labelled high-resolution images (ImageNet DS).

According to Mittal et al. [2] have CNNs already shown better predictive performance than traditional image processing or machine learning methods.

Luo et al. evaluate pre-processing, segmentation, and post processing for an accurate classification of 108 plankton classes. The authors use greyscale images, which are fraught with noise and unevenness in greyscale and contrast. In flat fielding, a calculated calibration frame is subtracted from the raw image. They use histogram normalization to normalize the contrast in each image, which allows for better segmentation of ROIs. Extremely noisy images, i.e. those with a signal-to-noise (SNR) below 25, are sorted out. Then use the K-harmonic mean clustering to detect and segment the ROIs.

Deep et al. [6] propose a CNN model and two CNN+shallow models for the classification of living fish species. In their dataset, most of the images are noise-free but out of focus. Therefore, the images are first preprocessed with an image sharpening method to improve the edges in the images. They are using a slightly modified Laplacian kernel, where the sum of all kernel elements is one instead of zero. This kernel produces a colour image, while the original Laplacian kernel produces a binary image.

## 3. Image Sets

The images set consists of different underwater images with a high variance in illumination conditions, spatial orientation, noise (bubbles, blurring), and colour palettes. The images are snapshots taken from videos recorded by a human diver. The images are used for supervised ML requiring explicit labelling. The labelling is done by hand by interactive drawing of labelled closed polygon paths assigning regions of the images to a specific class. There are areas remaining with no / unknown labelling.

## 4. Methods and Architecture

In addition to the evaluation of suitable algorithms and classification models, this work compares two different software frameworks:

1. Native software code using widely deployed TensorFlow-Keras with GPU support [7];
2. Pure JavaScript code using PSciLab with WokBooks and Workshells [1] (processed by a Web browser and node.js), using a customized version of the ConvNet.js trainer for CNNs.

The second framework stored all data in SQL data bases as well as trained models (JSON format). The SQL data bases (SQLite3) can be accessed remotely via a SQLjson Remote Procedure Call (RPC) interface. The TensorFlow framework used the local filesystem for data storage. Any computer processing TensoFlow needed a copy of the entire data sets.

Both software frameworks used the same input data and functionally and structural equivalent CNN architectures.

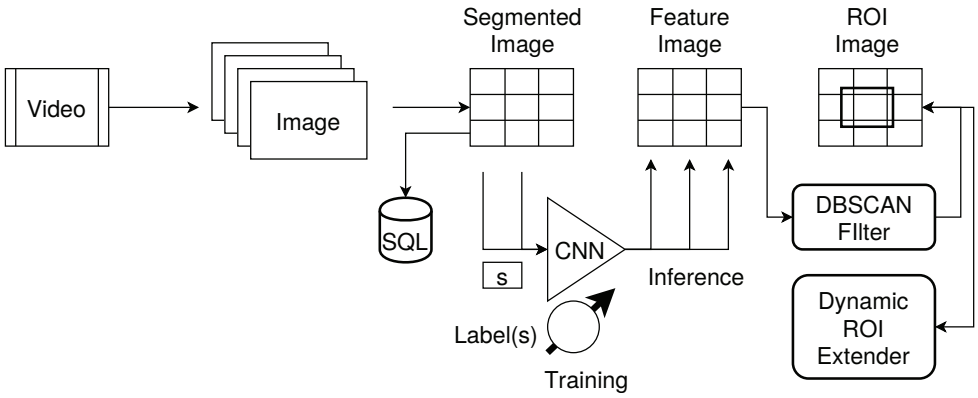



**Fig. 1.** Overview of the data flow architecture and the used algorithms

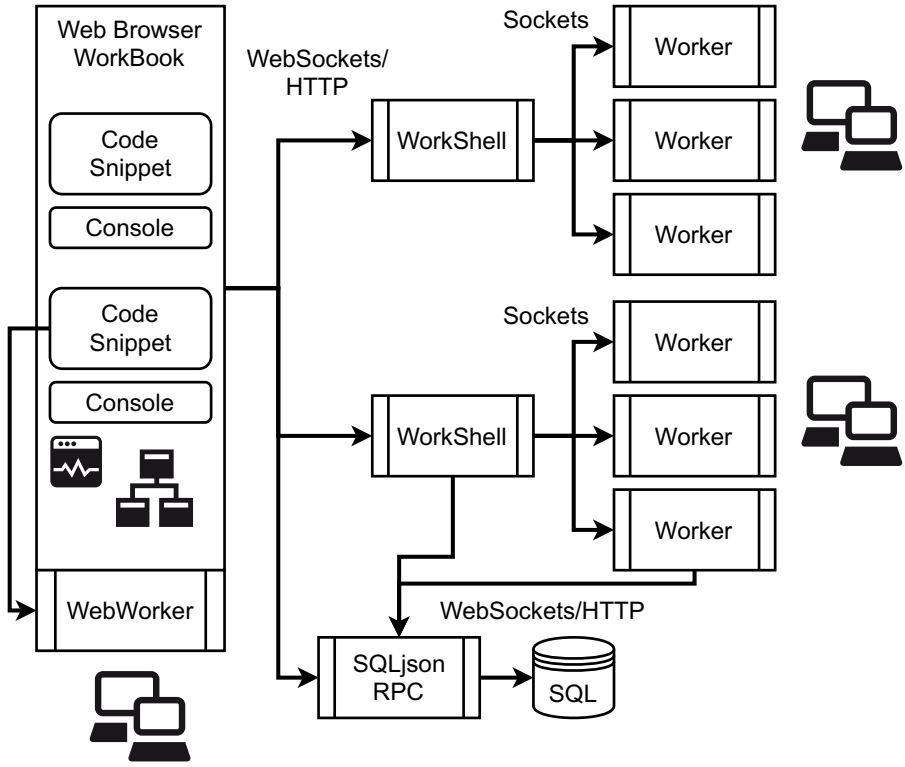



**Fig. 2.** Web browser-based software architecture with remote worker processes [1]

### *4.1 Image Segmentation*

On the first processing level the input images will be segmented in equally sized sub-images, e.g., RGB segments of 64 × 64 pixels. Each image segment is related to one of the classes $\sigma \in$ {B,P,C} or unknown (U). A conventional CNN with two convolutional layers is used to predict the class $\sigma \in$ {B,P,C} for each single image segment. The CNN is trained with a sub-set of randomly chosen labelled image segments.

### *4.2 Convolutional Neural Network Architecture*

Four different CNN architectures and parameter settings were evaluated, summarized in Tab. 1, assuming a segment input site data volumes with 64 × 64 × 3 (RGB) elements (derived from the RGB video images). Both software frameworks used the same CNN architecture and configuration. The smallest CNN model compared with the largest requires about 1/4 of the unit vector operations and about 1/3 of the hyper parameters that must be trained.

| Arch. | Layer | Filter | Activation | Output | Parameter | VecOps |
|---|---|---|---|---|---|---|
| A (16/16) | Conv | [5×5]×8,s=1 | - | 64×64×8 | 608 | 4915200 |
| | Relu | - | relu | 64×64×8 | 32768 | 32768 |
| | Pool | [2×2]×8,s=2 | - | 32×32×8 | 0 | 8192 |
| | Conv | [5×5]×16,s=1 | - | 32×32×16 | 3216 | 6553600 |
| | Relu | - | relu | 32×32×16 | 16384 | 16384 |
| | Pool | [3×3]×16,s=3 | - | 10×10×16 | 0 | 1600 |
| | Fc | - | relu | 1×1×3 | 4803 | 9600 |
| | SoftMax | - | - | 3 | 3 | 3 |
| | | | | | **Σ57782** | **Σ11537347** |
| B (8/8) | Conv | [5×5]×4,s=1 | - | 64×64×4 | 304 | 2457600 |
| | Relu | - | relu | 64×64×4 | 16384 | 16384 |
| | Pool | [2×2]×4,s=2 | - | 32×32×4 | 0 | 4096 |
| | Conv | [5×5]×8,s=1 | - | 32×32×8 | 808 | 1628400 |
| | Relu | - | relu | 32×32×8 | 8192 | 8192 |
| | Pool | [3×3]×8,s=3 | - | 10×10×8 | 0 | 800 |
| | Fc | - | relu | 1×1×3 | 2403 | 4800 |
| | SoftMax | - | - | 3 | 3 | 3 |
| | | | | | **Σ28094** | **Σ4127878** |
| C (8/16) | Conv | [5×5]×8,s=1 | - | 64×64×8 | 608 | 4915200 |
| | Relu | - | relu | 64×64×8 | 32768 | 32768 |
| | Pool | [2×2]×8,s=2 | - | 32×32×8 | 0 | 8192 |
| | Conv | [5×5]×16,s=1 | - | 32×32×8 | 1608 | 3276800 |
| | Relu | - | relu | 32×32×8 | 8192 | 8192 |
| | Pool | [3×3]×16,s=3 | - | 10×10×8 | 0 | 800 |
| | Fc | - | relu | 1×1×3 | 2403 | 4800 |
| | SoftMax | - | - | 3 | 3 | 3 |
| | | | | | **Σ45582** | **Σ8246755** |
| D (4/4) | Conv | [5×5]×4,s=1 | - | 64×64×4 | 304 | 2457600 |
| | Relu | - | relu | 64×64×4 | 16384 | 16384 |
| | Pool | [2×2]×4,s=2 | - | 32×32×4 | 0 | 4096 |
| | Conv | [5×5]×4,s=1 | - | 32×32×4 | 404 | 819200 |
| | Relu | - | relu | 32×32×4 | 4096 | 4096 |
| | Pool | [3×3]×4,s=3 | - | 10×10×4 | 0 | 400 |
| | Fc | - | relu | 1×1×3 | 1203 | 2400 |
| | SoftMax | - | - | 3 | 3 | 3 |
| | | | | | **Σ22394** | **Σ3304179** |

**Tab. 1.** Layer structure of and parameter count for four different CNN architectures used in this work (s:stride, vecOps: Unit vector operations, input layer has output size 32×32×3)

### *4.3 Image ROI Classification*

The basic algorithm and workflow for automated ROI classification:

1. Segmentation of each input image with static size segments;
2. Parallel prediction of the image segment class by the CNN;
3. Creation of a class prediction matrix $\hat{C}$ with rows and columns representing the spatial distribution of the image segments in the original input image; the matrix $M$ is considered as a point cloud with cartesian point coordinates related to the matrix ⟨row,column⟩ tuple;
4. Computation of spatial class element clusters using the *DBSCAN* algorithm; parameters 6epsilon; and *minPoints* must be chosen carefully (e.g., ε=2, *minPoints*=5);
5. Applying a Mean Bounding Box (MBB) algorithm to the point elements of each cluster computing the mass-centred average bounding box (typically under-sized with respect to the representative points in the clusters);
6. Applying an MBB extension iteratively to grow the bounding box but still suppressing spurious (wrong) image segments;
7. Remove small(er) bonding boxes covered by larger bounding boxes (either with different or same class) or shrink overlapping bounding boxes of different classes by priority decision (shrink less important regions);
8. Mark the original input image with ROI rectangles computed from the previous step;

Iteratively expanded bounding boxes from different classes can overlap, which is an undesired result. To reduce overlapping conflicts, a class priority is introduced. In this work, coverage on construction surfaces has the highest priority to be detected accurately. After the ROI expansion is done, overlapping bounding boxes with lower priority are shrink until all overlap conflicts are resolved.

### *4.4 Training and Labelling*

For training, a selected and representative sub-set of images (246 images) are extracted from the diving video. Each image is labelled manually by adding relevant and strong ROI polygons to each image. Based on the labelled and closed polygon paths, each image is segmented with a static segment size. All segments from an image are stored in a SQL database table. With respect to a given image size of $1920 \times 1080$ pixels, a chosen segment size of $64 \times 64$ pixels, there are about 120000 small labelled image segments. Segment images not covered by any of the labelled polygon paths are automatically marked with the class "Unknown". Only strong and clearly classifiable regions are created, shown in Fig. 3. Remaining unlabelled regions will not be considered for the training process.

The training process selects randomly a balanced sub-set of the image segments (e.g., 1000) with respect to the class label distribution, i.e., providing a normal distribution of the class labels among the training and validation data set. Multiple models are trained in parallel. Each model is trained with a different set of segments and with random initialisation of the model parameters using Monte Carlo simulation.

The TensorFlow framework used an Adam optimizer with a very low learning rate of 0.001. The ConvNetJS CNN framework used an adaptive gradient optimizer with a moderate learning rate of 0.1 and a high momentum of 0.9. Each convolution layer had an l2 regularization loss with l2=0.01 in TensorFlow framework and l2=0.001 in the ConvNetJS framework.

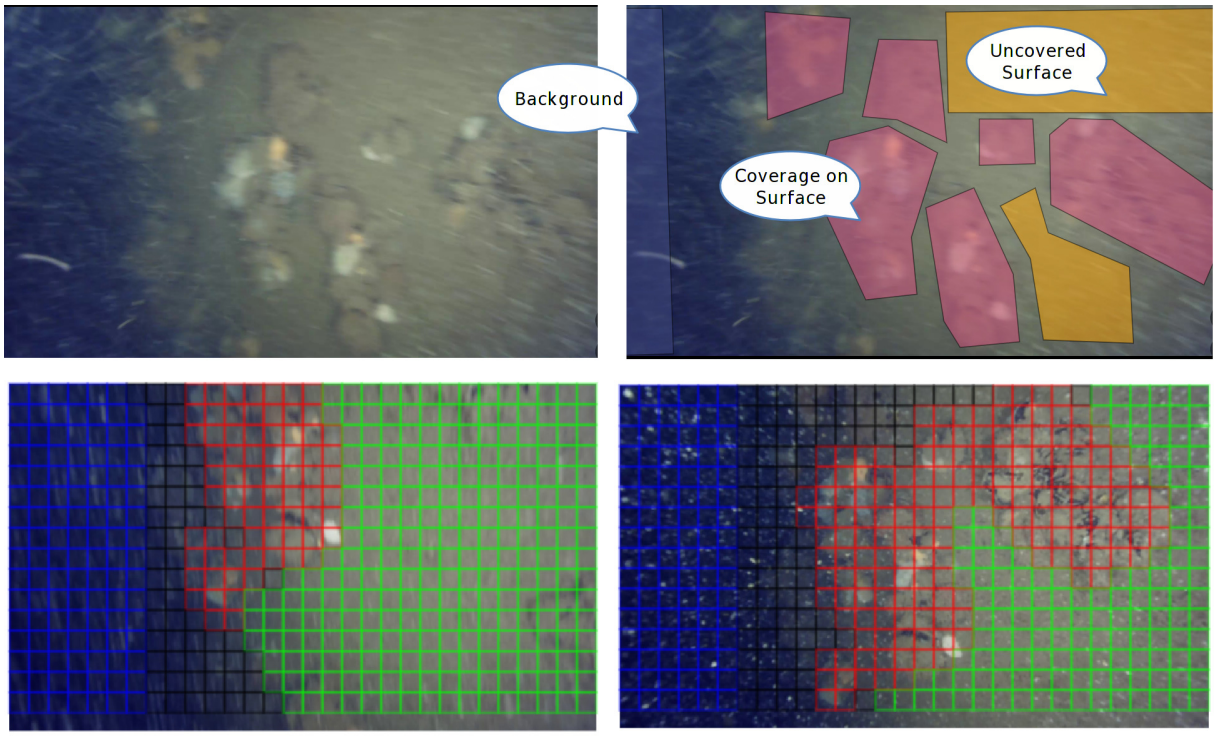



**Fig. 3.** Example of the manual labelling of polygon path bounded regions (Top, Left) Original image (Top, Right) With labelled polygon regions (Bottom) Segmented images

### *4.5 Mean Bounding Box Algorithm*

In this section the Mean Bounding Box (MBB) algorithm is introduced. There is a set of class symbols Σ

and a class matrix $\hat{M}$ consisting of elements labelling an image segment with a class, so that:

$$\begin{gathered}\Sigma = \{B, P, C, U\} \\ \sigma \in \Sigma \\ \hat{M} = \begin{pmatrix} \sigma_{1,1} & .. & \sigma_{1,j} \\ \sigma_{2,1} & .. & \sigma_{2,j} \\ .. & .. & .. \\ \sigma_{i,1} & .. & \sigma_{i,j} \end{pmatrix}\end{gathered} \tag{1}$$

The matrix $\hat{M}$ is flattened to a point cloud list set $P=\{p_\sigma\}_{\sigma\in\Sigma}$. Each class set $p$ contains the matrix positions of the respective elements, i.e., $p_\sigma=\{\langle i,j\rangle\}$, with all points classified by the CNN to the same label class $\sigma\in\Sigma$.

The DBSCAN clustering will return a group list of points that satisfy the clustering conditions, one point group list for each label class, as shown in Fig. 4 (a).

$$\begin{gathered}DBSCAN : P \rightarrow \left\{\{p_j\}_j, \{p_k\}_k, \{p_l\}_l, ..\right\}, j \neq k \neq l \\ P : \{p_i\}_i, i = \{1, 2, 3, .., n\} \\ p_i =< i, j >\in R^2\end{gathered} \tag{2}$$

It is assumed that a cluster will contain a majority of correctly classified points (segments), and a minority of scattered wrong classified points.

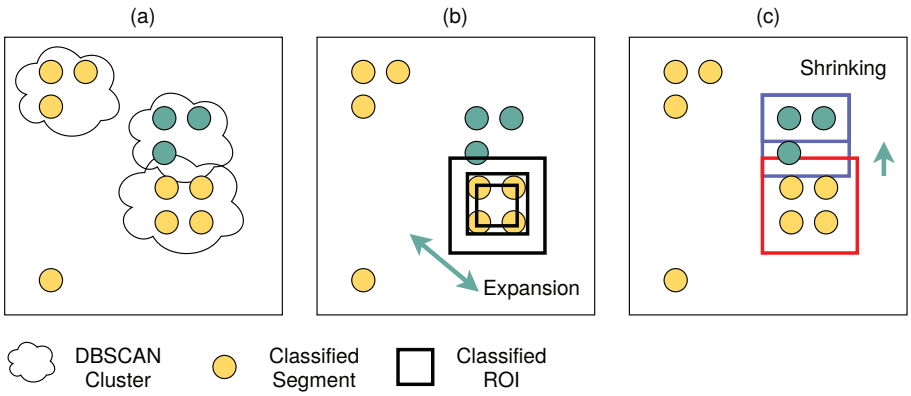



**Fig. 4.** Iterative bounding box expansion with final conflict overlapping shrinking

The MBB algorithm computes points $\langle x_1, y_1, x_2, y_2\rangle$ of a bounding box that is centred at the mass-of-centre point $c$ of all points of a cluster and with outer sides given by the vectorial mean centred position of all points above or below, and left or right form the $c$ point, as shown in Alg. 1 and in Fig. 4 (b).

```
 1: function massOfCenter(points)
 2:   pc = { x=0, y=0 }
 3:   ∀p ∈ points do
 4:     pc.x := pc.x+p.x, pc.y := pc.y+p.y
 5:   done
 6:   pc := pc / |points|
 7:   return pc
 8: end
 9: function meanBBox(points)
10:   pc = massOfCenter(points)
11:   // Initial bbox around mass-of-centre point
12:   b = { x1=pc.x, y1=pc.y, x2=pc.x, y2=pc.y }
13:   c = { x1=1, y1=1, x2=1, y2=1 }
14:   ∀p ∈ points do
15:     // each point extends the bbox
16:     if p.x>pc.x then incr(c.x2), b.x2 := b.x2+p.x
17:     if p.x<pc.x then incr(c.x1), b.x1 := b.x1+p.x
18:     if p.y>pc.y then incr(c.y2), b.y2 := b.y2+p.y
19:     if p.y<pc.y then incr(c.y1), b.y1 := b.y1+p.y
20:   done
21:   // normalize bbox coordinates
22:   b.x1 := b.x1 / c.x1, b.x2 := b.x2 / c.x2
23:   b.y1 := b.y1 / c.y1, b.y2 := b.y2 / c.y2
24:   return b
25: end
```

**Alg. 1.** Mean Bounding Box Algorithm applied to a two-dimensional point cloud

The expansion of a previously computed bounding box is done by all points outside of the current bounding box, performing the next extension iteration. Again, a spatial position averaging is performed, extending the boundary of the bound box, shown in Alg. 2. The expansion is performed iteratively. Each step includes more points, but increases the probability that the bound box is over-sized with respect to spurious outlier points that result from wrong CNN classifications.

```
 1: function meanBBoxExpand(points,b)
 2:   pc = massOfCenter(points)
 3:   // start with the old bbox
 4:   b2 = { x1=b.x, y1=b.y, x2=b.x, y2=b.y }
```

```
 5:   c = { x1=1, y1=1, x2=1, y2=1 }
 6:   ∀p ∈ points do
 7:     // each point outside the old bbox extends the new bbox
 8:     if p.x>b.x then incr(c.x2), b2.x2 := b2.x2+p.x
 9:     if p.x<b.x then incr(c.x1), b2.x1 := b2.x1+p.x
10:     if p.y>b.y then incr(c.y2), b2.y2 := b2.y2+p.y
11:     if p.y<b.y then incr(c.y1), b2.y1 := b2.y1+p.y
12:   done
13:   // normalize bbox coordinates
14:   b2.x1 := b2.x1 / c.x1, b2.x2 := b2.x2 / c.x2
15:   b2.y1 := b2.y1 / c.y1, b2.y2 := b2.y2 / c.y2
16:   return b2
17: end
```

**Alg. 2.** Mean Bounding Box expansion applied to a two-dimensional point cloud and mean bound box

In case of high iteration loop values, bounding boxes from different classes can overlap. To reduce overlapping conflicts, a class priority is introduced layering the class regions by relevance. After the ROI expansion is done, overlapping bounding boxes with lower priority are shrink until all overlap conflicts are resolved. Commonly, more than one side of the bounding box can be shrunken to reduce the overlap conflict. The possible candidates are evaluated and sorted with respect to the amount of shrinkage at each side. The lowest shrinkage is applied first. If the conflict is not reduced by the selected side shrinking, the next side is shrink until the conflict (with one or more higher priority bounding boxes) is reduced, as shown in Fig. 4 (c).

## 5. Results

The original numeric loss computed from the softmax layer and returned by the trainer is not a measure for the discrete prediction accuracy, i.e., the number of correct and incorrect predicted segment classes, which are achieved after binarization and maximum best-of selection. This is an indicator for a low separation margin in the target feature space. There is no significant difference in the accuracy, recall, and precision in the training and test data set, shown in Tab. 2. Examples of classified bounding boxes are shown in Fig. 5. Because only the C class (coverage of construction surface) is of high relevance (the highest priority), only the particular classification percentages for this class are shown in the last column in Tab. 2. The average prediction error for all classes is about 10% with low variance across different models trained with different sub-sets from the entire data space and each with different random initialisation. The average error for specific classes differ significantly. The relevant class C shows a prediction error (¬ C) about 20% with respect to samples and a high variance across different models. Splitting the prediction accuracy in the tuple true positive (C), false positive (¬ C), true negative (¬ C), and false negative (C), the average TP prediction accuracy is about 80%.

We get the following average statistical measures for the class prediction of single image segments:

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{FN}+\text{TN}} = \begin{cases} 0.92 & \text{train} \\ 0.91 & \text{test} \\ 0.92 & \text{all} \end{cases} \\
\text{Precision} &= \frac{\text{TP}}{\text{TP}+\text{FP}} = \begin{cases} 0.94 & \text{train} \\ 0.94 & \text{test} \\ 0.94 & \text{all} \end{cases} \\
\text{Recall} &= \frac{\text{TP}}{\text{TP}+\text{FN}} = \begin{cases} 0.88 & \text{train} \\ 0.88 & \text{test} \\ 0.88 & \text{all} \end{cases} \\
\text{Specificity} &= \frac{\text{TN}}{\text{TN}+\text{FP}} = \begin{cases} 0.95 & \text{train} \\ 0.95 & \text{test} \\ 0.95 & \text{all} \end{cases} \\
f_1 &= 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall}+\text{Precision}} = \begin{cases} 0.9 & \text{test} \\ 0.9 & \text{train} \\ 0.9 & \text{all} \end{cases}
\end{aligned} \tag{3}$$

Prediction results for training and test data do not differ significantly and show similar high statistical measures, which is an indicator for a representative training data sub-set and a sufficiently generalised predictor model.

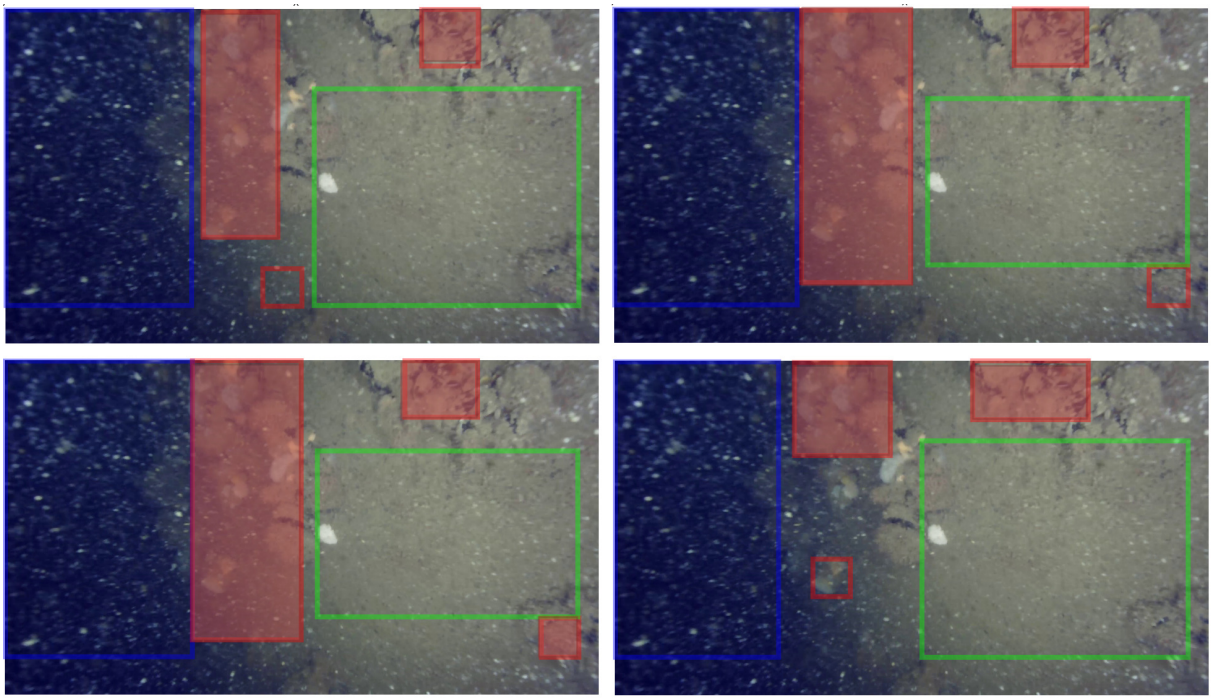


**Fig. 5.** Classified bounding boxes for one image using four models trained in parallel (same parameters) but with different random initialisation and training data sub-set (Blue: class background, red: class coverage, green: class free construction surface)

Considering the bounding box estimator post-processing, the FP rate of the priority class C is nearly zero. The average coverage of the predicted and estimated C area is about 50%, showing an underestimation. The TP rate of class C regions is about 70%.

| **Data Set** | **Total error ($\neg TP_C$) %** | **Error ($\neg TP_C$)/class %** | **Prediction accuracy/Class C (TP,FP,TN,FN) %** |
|---|---|---|---|
| Training | 10.6±1.5 | 5.0±3.4,6.0±2.8,21.0±7.1 | 79.0±7, 4.8±2, 94.7±6.6, 10.5±3.1 |
| Test | 11.1±1.8 | 5.3±2.6, 5.8±3.2, 22.0±8.3 | 78.0±4.3, 5.1±2.2, 95.1±2.1, 11.0±4.4 |
| All | 10.9±1.6 | 4.2±2.8, 5.9±3.4, 21.7±8 | 78.4±8, 5.0±2.2, 95.0±2.2, 10.8±4 |

**Tab. 2.** Accumulated prediction results for training, test, entire data set union with statistical features of the model ensemble trained in parallel (using different data sub-sets and random initialisation). All errors with $2\sigma$ standard deviation interval, and N=9000 samples, n=3000 for each class, and using CNN architecture A.

| **CNN Architecture** | **Parameters** | **Forward Time** | **Backward Time** |
|---|---|---|---|
| A (8/16) | 122587 | 18 ms[1], 0.5 ms[2] | 26 ms[1], 1 ms[2] |
| B (4/8) | 66639 | 8 ms[1] | 10 ms[1] |
| C (8/8) | 104603 | 12 ms[1] | 18 ms[1] |
| D (4/4) | 58047 | 6 ms[1] | 8 ms[1] |

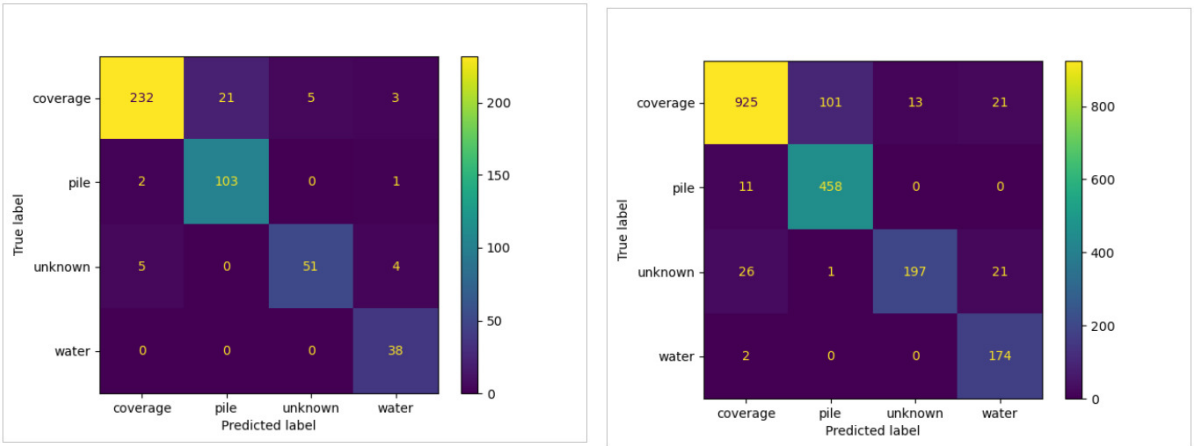
**Tab. 3.** Forward and backward (training) times for one $64 \times 64 \times 3$ segment and different CNN architectures (see Fig. 1) using the JavaScript ConvNet.js classifier[1] and TensorFlow (CPU)[2]

Finally, the different CNN architectures are compared with respect to classification accuracy in Tab. 4. There is no significant degradation of the classification accuracy observed.

| CNN | Total Error % | Accu | Prec | Recall | Spec | f1 |
|---|---|---|---|---|---|---|
| A (8/16) | 11.8 | 0.909 | 0.935 | 0.866 | 0.947 | 0.899 |
| B (4/8) | 11.8 | 0.909 | 0.935 | 0.866 | 0.947 | 0.899 |
| C (8/8) | 11.7 | 0.909 | 0.936 | 0.864 | 0.948 | 0.899 |
| D (4/4) | 12.7 | 0.900 | 0.924 | 0.856 | 0.938 | 0.889 |

**Tab. 4.** Statistical measures for the different CNN model architectures

In addition to a three-class predictor, a four-class predictor was evaluated, too. An arbitrary unknown class *U* was added to the class set (i.e., a void class covering "all other" cases). There were no significant improvement in prediction accuracy of the classes B/P/C observed. A confusion matrix plot of a image segment classification example is shown in Fig. 6. Reducing the image segment size by a factor 2 increase classification errors significantly, suggesting the 64 × 64 segment size as an lower limit.




(a) 64x64 pixels

(b) 32x32 pixels

**Fig. 6.** Example results (from TensorFlow model) of a four-class predictor with an additional "unknown" class U (a) 64 × 64 pixel segment size (b) 32 × 32 pixel segment size

Increase in accuracy after some image pre-processing techniques (Gaussian blur, rotation) is small and would be around 1-2%. One major question is the explainability of the CNN classifier and which features of the input image segments are amplified. A first guess was the colour information contained in the image segments, i.e., background is mostly blue or black, coverage is mostly white or grey. Therefore, a simple RGB-pixel classifier was applied to each image pixel using a simple fully connected ANN, finally applying the same post-processing algorithms. The result showed an average true-positive classification accuracy of about 60%, which is above guess likelihood (33%), and therefore the colour feature is still strongly correlated to the classification target label.

Comparing both software frameworks (optimized native code versa virtual machine processing with JavaScript), the overall classification results are similar, the overall segment classification accuracy is about 90%. The computational time of ConvNet.js is about 50 times higher than the CPU-based TensorFlow software. Because the CNN complexity is low (less than 100000 parameters distributed over 6 layers), data-path parallelization using Single-Instruction Multiple-Data architectures and GGPU co-processors pose no significant speed-up. Control-path parallelization can be utilised during training of the model ensemble (maximal speed-up $M$ with $M$ models), and during inference (maximal speed-up is $S$ with $S$ as the number of segments per image).

| **Model (Input size)** | **Raspberry Pi-3/4 (TF-Lite)** | **Raspberry Pi-3/4 Intel Neural Stick 2** | **Raspberry Pi-3/4 Google Coral USB** | **Jetson Nano** | **Google Coral** |
|---|---|---|---|---|---|
| EfficientNet-B0 (224x224) | 14.6-25.8 FPS | 95-180 FPS | 105-200 FPS | 216 FPS | 200 FPS |
| ResNet-50 (244x244) | 2.4-4.3 FPS | 16-60 FPS | 10-18.8 FPS | 36 FPS | 18.8 FPS |
| MobileNet-v2 (300x300) | 8.5-15.3 FPS | 30 FPS(Pi-3) | 46 FPS(Pi-3) | 64 FPS | 130 FPS |
| SSD Mobilenet-V2 (300x300) | 7.3-13 FPS | 11-41 FPS | 17-55 FPS | 39 FPS | 48 FPS |

**Tab. 5.** TensorFlow performance using widely used image classification networks processed on different hardware in image frames per second (FPS) Source: [7]

There are different choices for accelerated co-processors, but some of them are limited to TensorFlow only (proprietary interface). The Intel Neural Stick and the Google Coral accelerator are USB dongles with a special TPU chip performing all tensor calculations. The Google Coral works with special pre-compiled Tensor-

Flow Lite networks. The Jetson Nano is the only single-board computer with floating-point GPU acceleration. It supports most models because all frameworks such as TensorFlow, Caffe, PyTorch, YOLO, MXNet, and others use the CUDA GPU support library at a given time. The Raspberry Pi computer can be used with some computational accelerators- Intel Neural Stick2, Google coral USB accelerator. Google Coral development board has TPU(Tensor Processing Unit) in itself. Jetson Nano has GPU on board. TensorFlow Lite is compatible with all devices. Originally developed to work in smartphones and other small devices, TensorFlow Lite would never meet a CUDA GPU. Hence, it does not support CUDA or cuDNN. So, Usage of TensorFlow Lite on Jetson Nano is purely based on CPU, not with GPU. Jetson Nano can run TensorFlow models with GPU on board. But NVIDIA (Jetson is from NVIDIA) provides TF-TRT on Jetson Nano. TensorFlow-TensorRT (TF-TRT) is an integration of TensorFlow and TensorRT that leverages inference optimization on NVIDIA GPUs within the TensorFlow ecosystem.

Tab. 5 shows a summary of TensorFlow performance using widely used image classification networks and processed on different hardware devices using accelerators.

## 6. Conclusion

Although the overall classification accuracy is about 90%, the high variance of the segment prediction results across differently trained models (model ensemble all having the same architecture) limits the output quality of the labelled ROI detector, typically resulting in an underestimation of the classified regions and a lacking of generalisation. But the presented static segment prediction with point clustering and iterative selective bounding box approximation with final overlap conflict reduction is still reliable. Similar to random forest trees, a multi-model prediction with model fusion (e.g., major coverage estimation) is proposed to get the best matching bonding boxes for the relevant classes.

The reduction of the CNN complexity with respect to the number of filters and dynamic parameters does not lower the classification accuracy significantly. Although, CNN are less suitable for low-resource embedded systems, the CNN architecture D (4/4) could be implemented in an embedded camera systems, expecting overall ROI extraction times for one image frame about 5 seconds, not suitable for real-time operation (maximal latency 100 ms). Using control-path parallelisation performing the image segment classifications in parallel, the ROI extraction could be reduced to 1 second using generic multi-core CPUs, or 100 ms using FPGA-based co-processors.

**Founding**

This work was founded by the Bremer Aufbau-Bank GmbH, FUE0648B, for the project "Maritime KI unterstützte Bildauswertung" (MaritimKIB).

## 7. References

[1] Bosse, S., PSciLab: An Unified Distributed and Parallel Software Framework for Data Analysis, Simulation and Machine Learning—Design Practice, Software Architecture, and User Experience , Appl. Sci. 2022, 12(6), 2887; 10.3390/app12062887

[2] Mittal, Sparsh; Srishti Srivastava, J Phani Jayanth. 2021. A Survey of Deep Learning Techniques for Underwater Image Classification. DOI: 10.13140/RG.2.2.25098.59846

[3] Li, C., Anwar, S., Hou, J., Cong, R., Guo, C., & Ren, W. (2021). Underwater image enhancement via medium transmission-guided multi-color space embedding. IEEE Transactions on Image Processing, 30, 4985-5000.

[4] Akkaynak, D., & Treibitz, T. (2019). Sea-thru: A method for removing water from underwater images. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 1682-1691)

[5] Xu, Y. Zhang, H. Wang and X. Liu, Underwater image classification using deep convolutional neural networks and data augmentation, 2017 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), 2017, pp. 1-5, doi: 10.1109/ICSPCC.2017.8242527.

[6] Deep, B Vikram und Ratnakar Dash. Underwater fish species recognition using deep learning techniques. In Intl. Conf. on Signal Processing and Integrated Networks (SPIN), pages 665–669, 2019

[7] https://qengineering.eu/deep-learning-with-raspberry-pi-and-alternatives.html, on-line, accessed 1.7.2022