Iterative, Deep Synthetic Aperture Sonar Image Segmentation

Yung-Chen Sun®, Isaac D. Gerg®, and Vishal Monga®, Senior Member, IEEE

Abstract—Synthetic aperture sonar (SAS) systems produce high-resolution images of the seabed environment. Moreover, deep learning has demonstrated superior ability in finding robust features for automating imagery analysis. However, the success of deep learning is conditioned on having lots of labeled training data but obtaining generous pixel-level annotations of SAS imagery is often practically infeasible. This challenge has thus far limited the adoption of deep learning methods for SAS segmentation. Algorithms exist to segment SAS imagery in an unsupervised manner, but they lack the benefit of state-of-the-art learning methods and the results present significant room for improvement. In view of the above, we propose a new iterative algorithm for unsupervised SAS image segmentation combining superpixel formation, deep learning, and traditional clustering methods. We call our method iterative deep unsupervised segmentation (IDUS). IDUS is an unsupervised learning framework that can be divided into four main steps: 1) a deep network estimates class assignments; 2) low-level image features from the deep network are clustered into superpixels; 3) superpixels are clustered into class assignments (which we call pseudo-labels) using k-means; and 4) resulting pseudo-labels are used for loss backpropagation of the deep network prediction. These four steps are performed iteratively until convergence. A comparison of IDUS to current state-of-the-art methods on a realistic benchmark dataset for SAS image segmentation demonstrates the benefits of our proposal even as the IDUS incurs a much lower computational burden during inference (actual labeling of a test image). Because our design combines merits of classical superpixel methods with deep learning, practically we demonstrate a very significant benefit in terms of reduced selection bias, i.e., IDUS shows markedly improved robustness against the choice of training images. Finally, we also develop a semi-supervised (SS) extension of IDUS called Iterative Deep SS Segmentation (IDSS) and demonstrate experimentally that it can further enhance performance while outperforming supervised alternatives that exploit the same labeled training imagery.

Index Terms—Deep learning, image segmentation, semisupervised learning, synthetic aperture sonar (SAS), unsupervised learning.

I. INTRODUCTION

SYNTHETIC aperture sonar (SAS) is a coherent imaging modality which is capable of producing high-resolution images of the seabed environment [8]–[11] and constant-resolution images of the seafloor. This is an improvement over traditional side-scan-sonar (SSS) in which the resolution is a function of range. Both theoretical and practical underpinnings of SAS imaging have been well established over the last few decades [1]–[6]. Currently, SAS systems deployed on autonomous underwater vehicles (AUVs) [7] produce high-resolution images containing rich seabed information. The segmentation of these images plays an important role in analyzing the seabed environment. Fig. 1 shows examples of SAS images containing complex seabed textures. Accurate segmentation of SAS imagery provides several benefits including large-scale environmental understanding of the seabed [8]–[11] and in situ image interpretation to support AUV operations [12].

One challenge faced by SAS seabed segmentation is acquiring labeled data. Obtaining pixel-level labels is time-consuming and often requires a diver survey to obtain correct results. This lack of labeled data makes it difficult to deploy convolutional neural networks (CNNs) for SAS seabed segmentation as such methods usually require lots of labeled data to work well.

Many algorithms have been proposed to avoid the demand for labeled data [13]–[16]. However, all of them are based on hand-crafted features which means they cannot jointly optimize feature extraction and clustering as the features are fixed. Some unsupervised image classification works [17], [18] have shown that embedding deep learning steps into a clustering algorithm is a promising direction.

In this work, we propose a CNN-based technique which overcomes the need to train on large datasets of pixel-level labeled data to achieve success. Specifically, we note the correlation between class boundaries of superpixels and human notated segmentation of SAS images [19]. We devise a novel scheme which combines classical superpixel segmentation...
algorithms (which are non-differentiable) with deep learning (which is differentiable) to obtain a deep learning algorithm capable of using superpixel information of unlabeled pixels as a way to overcome the lack of abundant training data.

We evaluate our method, which we call iterative deep unsupervised segmentation (IDUS), against several comparison methods on a contemporary real-world SAS dataset. We show compelling performance of IDUS against existing state-of-the-art methods for SAS image co-segmentation. Additionally, we show an extension of our method which can obtain even better results when semi-labeled data is present. Specifically, our work makes the following contributions.

1) We devise a new iterative method to integrate the merits of best-known unsupervised superpixel segmentation along with a deep feature extractor. In particular, we utilize superpixels to generalize image-level clustering to superpixel clustering and propose an unsupervised learning framework for training a U-Net-based CNN [20] in image segmentation tasks. Our proposal exceeds state-of-the-art performance on the task of unsupervised segmentation for seabed sonar images.

2) We propose a supervised extension for IDUS, which called Iterative Deep Semi-supervised Segmentation (IDSS), to increase network performance in situations that do not have enough annotated data.

3) IDUS is considerably faster than the previously widely used methods, which encourages the deployment of IDUS on applications that require real-time speed.

4) Our approach does not depend on any specific network structure or superpixel generation algorithm. The user can adjust the algorithm easily according to different segmentation tasks.

Reproducibility: To facilitate verification of results reported in this article as well as provide a resource for the SAS segmentation research community, we make our code and trained network available at: https://scholarsphere.psu.edu/resources/ff521a5e-58e8-48b2-a9c9-5012800b62ab

II. RELATED WORK

A. Seabed Texture Segmentation

Various unsupervised feature extraction methods related to our have been proposed. Williams et al. [14] presented a sand ripple model using sinusoid functions with different orientations and scales to characterize the sand ripple pattern as three highlight-shadow pairs. Cobb et al. [21] applied autocorrelation function to extract seabed features which perform especially well for extracting periodically patterns. However, it has been reported that autocorrelation is not a suitable measure for coarseness [22] (i.e., if a texture has bigger element sizes and fewer elements are repeated, the texture is coarser [23]). Some works [16], [21] argue that single point statistics (such as gamma distribution variance or K-distribution shape parameter) are useful descriptors to represent seabed environments and easily differentiate between smooth and coarse textures. Williams [24] uses the coefficient of multi-scale orthogonal wavelet decomposition to extract the features where each sliding window of SAS images is treated as a unique data point. Samiee et al. [25] and Cobb et al. [26] used the superposition of Gabor filter banks response to represent seabed textures.

In traditional methods based on hand-designed features, the image patches usually are fed to a series of filter banks using sliding windows. Next, superpixels are used to quantize pixel-level features into regional features, and finally applied some clustering techniques such as k-means or Bayesian mixture model [27] to cluster these superpixels. Some works such as [16] and [26] would additionally use the idea of texton [28] to cluster the features into groups to find the prototypes of features before performing superpixel quantization. However, these kinds of hand-craft-based features are not differentiable and thus cannot be end-to-end trained. Therefore, the parameters of feature extraction are fixed and cannot be jointly optimized with the classifier.

There has been recent work in deep-networks [29], [30] trained for sidescan sonar images to segment seabed environment textures. These models are trained in a purely supervised fashion (using tedious human annotation), with training and evaluation over a relatively homogeneous image collection. Hence, their generalization ability is difficult to ascertain.

B. Unsupervised Deep Feature Learning

Many works have been proposed for deep feature learning without any human-annotated labels and are generally summarized as self-supervised or clustering-based methods.

1) Self-Supervised Methods: For self-supervised learning, pretext tasks [31] are pre-designed for networks to solve, and some characteristics of the dataset are utilized to generate the pretext task goals (i.e., ground truth for pretext tasks) computing the loss function. In other words, visual features are learned by learning objective functions of pretext tasks. The commonly used pretext task includes generation-based methods such as image colorization [32], image super-resolution [33], image inpainting [34], image generation by generative adversarial networks (GANs) [35], [36]. Some pretext tasks hidden spatial information and let the CNN correctly restore it, includes image jigsaw puzzle [37]–[40], context prediction [41], and geometric transformation recognition [36], [42]. Some pretext tasks are designed to learn visual features from videos providing rich temporal information. These tasks disorder the video frames first and force a CNN to predict if the input frame sequence is correct [43], [44] or the correct order of frame sequence [45], [46]. Although pretext-based methods can learn discriminative features, those approaches usually require experts that have the domain knowledge to carefully design. Therefore, it is not easy to transfer these approaches to other domains.

2) Clustering-Based Methods: The combination of clustering techniques and CNNs has recently been successful on small datasets in that it can go largely beyond the performance of traditional frameworks, i.e., performs clustering without updating features [47]–[50]. For large-scale datasets such as ImageNet [51] and DeepCluster [17] provided an easily implemented solution using k-means to cluster the deep features and reassign new pseudo-labels to each image. This
makes unsupervised learning of the features on large-scale datasets be possible. Zhan et al. [18] solves the instability problem of DeepCluster by breaking down the global cluster into the batch-wise deep features clustering and labels update. These developments motivate us to develop our own work in unsupervised image segmentation wherein we will seek to integrate domain insight of unsupervised superpixel-based sonar/SAS segmentation methods with the modeling power of deep learning.

III. Method

A. Background and Notation

Given a set $X = \{x_1, \ldots, x_N\}$ of $N$ SAS images, $s^k_n$ is the $k$th superpixel of the $n$th image and $r^k_n$ are the pseudo-label of this superpixel. We use $h_\theta(\cdot)$ to denote the softmax output of a segmentation network (i.e., deep network), where $\theta$ is the parameters we need to optimize. The output has the same size with the input image. Given an input image $x_n$, the network maps the image into a feature map $f_n = h_\theta(x_n)$ in which $f^i_n$ represent the feature of $i$th pixel $x^i_n$ in this image.

1) Superpixel Pooling: To quantize the pixel-level features into region-level features $z^k_n = Q_{pool}(x_n, f^i_n, s_n)$, we compute the feature’s mean from all the pixels making up the superpixel $s^k_n$ by the following equation:

$$z^k_n = \frac{\sum_{i=1}^{D} f^i_n I(x^i_n \in s^k_n)}{\sum_{i=1}^{D} I(x^i_n \in s^k_n)}$$

(1)

where $D$ denotes the number of pixels in the input image. $I(x^i_n \in s^k_n)$ is an indicator function that is 1 if $x^i_n$ belongs to the superpixel $s^k_n$. The superpixel quantization makes sure that we can collect regional features of sufficient accuracy for the later clustering use.

2) Superpixel Mapping: Suppose we have already known a superpixel label $r^k_n$, then every pixel label $y^i_n$ in this superpixel should be the same. This mapping is given by

$$y^i_n = \sum_{k=1}^{K_n} r^k_n I(s^i_n \in s^k_n)$$

(2)

where $K_n$ denotes the number of superpixels in $n$th image. We use $y_n = Q_{map}(x_n, f_n, s_n)$ to denote this process which we call superpixel mapping. It eliminates some classification errors and produce more robust class assignments for pixels.

B. Iterative Deep Unsupervised Segmentation

DeepCluster [17] alternates between updating the model’s parameters and reassigning pseudo-labels by $k$-means to the input images. Every image is mapped to a deep feature; however, this process loses boundary information between different textures. In other words, the deep feature can only represent a region with a rough boundary which is not accurate enough for unsupervised image segmentation. Different from DeepCluster, we use the idea of a superpixel to make sure that we can use the deep features to represent regions with accurate boundaries and allow learning of a segmentation network without any human-annotated pixel-labels. To better understand the proposed idea, we introduce the simplest version of IDUS first and then a complete description will follow.

1) Simplest Version of IDUS: Fig. 2 shows an iteration of the simplest version of IDUS after initialization (will be introduced in Section III-D1). Here, we assume that the dataset only contains one image. We store the superpixel segments and labels in the superpixel segments memory and superpixel labels memory, respectively, since we will not update the segments and labels every training epoch for the consideration of computation efficiency. We see the softmax vectors $f_n$ of network outputs as a kind of feature representation due to it has the same size of the inputs and contains the distance information between a region texture and cluster centroids. Given the feature map (pixel features) $f_n$ ingesting from the input image via a segmentation network, an iteration of IDUS contains six main steps.

1) The pixel features are fed into simple linear iterative clustering (SLIC) [52] to generate new superpixel segments and replace the original ones.

2) The pixel features are quantized into superpixel features [by(1)] with the new segments.
3) The superpixel features are clustered by k-means to update the class assignments in memory.
4) The superpixel labels are mapped to pseudo pixel labels via superpixel mapping (2) with the new segments.
5) A loss is computed via the pseudo pixel labels and the network softmax outputs.
6) The network is updated via backpropagation.

2) Complete Version of IDUS: More precisely, given a set of training images \( X_T = \{x_n\}_{n=1}^{N_T} \), we update the model’s parameters \( \theta \) by solving the following problem:

\[
\min_{\theta} \frac{1}{N_T} \sum_{n=1}^{N_T} \ell(h_\theta(x_n), Q_{\text{map}}(x_n, r_n, s_n))
\]  

(3)

where \( \ell \) is a loss function and \( r_n, s_n \) will be initialized before performing the algorithm. After training the model for \( U_E \) epochs, we extract the feature maps \( \{f_n\}_{n=1}^{N_T} \) of all training images based on the current model’s parameters \( \theta_o \), where \( \omega \) is the number of iterations. Then, we pool the feature maps to regional features \( \{z_n\}_{n=1}^{N_T} \) and cluster them into \( M \) groups based on the optimization goal

\[
\min_{c_m \in \mathbb{R}^{d \times K}, \{z_n\}_{n=1}^{N_T}, \{I(k_n \equiv m)\}_{n=1}^{N_T}} \sum_{m=1}^{M} \sum_{n=1}^{N_T} \sum_{k=1}^{K} \min_{s_d \in [1, M]} \left\| z_n^k - c_m \right\| I(k_n \equiv m)
\]  

(4)

where \( c_m \) is the centroid feature of class \( m \) and \( d \) is the length of feature dimension. With this clustering criterion, the regional features will be clustered and used to produce new labels \( \{r_n\}_{n=1}^{N_T} \); and then, they will be assigned to the corresponding superpixels.

The overall idea of IDUS can be summarized in Algorithm 1. We update the superpixel boundaries every \( U_S \) epochs and the algorithm terminates after a fixed number of iterations. Note, despite updating the superpixels based on the intermediate representation of the network, the loss function is still differentiable with respect to the input and network parameters. The network outputs, \( f_n \), are only needed during the back-propagation step (and hence not used during the forward pass) during training. Consequently, the superpixel clustering performed on \( f_n \) is implicitly learned during training while still allowing the objective function to be differentiable. This is the same approach employed by the DeepCluster [17] algorithm on which our method is based.

C. Iterative Deep Semi-Supervised Segmentation

Although pixel-level labels are expensive to collect, it is not uncommon to see the “easy” areas of the images labeled by human operators. To utilize this extra information, we propose IDSS to combine IDUS with (semi-)supervised training and improve the network performance. Specifically, we use \( \{X_{\text{unlabel}}\} \) to denote the unlabeled dataset and \( \{X_{\text{label}}, Y_{\text{label}}\} \) the dataset where \( Y_{\text{label}} \) is the corresponding ground truth. To combine IDUS with supervised training, we first use \( \{X_{\text{unlabel}}, X_{\text{label}}\} \) to learn the visual features of U-Net by IDUS. Then, we extract the U-Net and re-initialize its segmentation head (linear classifier after the decoder). Finally, the pre-trained U-Net is fine-tuned on the labeled dataset \( \{X_{\text{label}}, Y_{\text{label}}\} \) which has fewer samples. We describe the exact design of IDSS in Algorithm 2.

Algorithm 1 IDUS

1: Initialize superpixel segments, labels \( \{s_n, r_n\}_{n=1}^{N_T} \) of \( N_T \) training images. And map the regional labels to pseudo pixel labels \( \{y_n\}_{n=1}^{N_T} \) by Eq (2).
2: Assign intervals \( U_E, U_S \) for updating the pseudo-ground-truth and superpixel boundaries.
3: Set the iteration index \( \omega = 0 \) and assign a number for maximum iterations \( \Omega \).
4: repeat
5: Given training dataset \( \{X_n, y_n\}_{n=1}^{N_T} \), update the model’s parameters \( \theta_o \) based on the problem Eq (3).
6: if every \( U_S \) epochs then
7: Update superpixel boundaries \( \{s_n\}_{n=1}^{N_T} \) based on the model of current iteration.
8: end if
9: if every \( U_E \) epochs then
10: Use the model parameters of current iteration \( \theta_o \) to generate pixel-level features \( \{f_n\}_{n=1}^{N_T} \).
11: Pool the pixel-level features into regional features (superpixel features) by Eq (1).
12: Cluster the regional features based on criterion Eq (4) and reassign new pseudo labels \( \{r_n\}_{n=1}^{N_T} \) to each superpixel.
13: end if
14: \( \omega \leftarrow \omega + 1 \).
15: until \( \omega \geq \Omega \).
16: Output the model’s parameters \( \theta_{\Omega} \).

Algorithm 2 IDSS

1: Input the unlabeled dataset \( \{X_{\text{unlabel}}\} \) and labeled dataset \( \{X_{\text{label}}, Y_{\text{label}}\} \).
2: Use \( X_{\text{unlabel}} \) and \( X_{\text{label}} \) to learn feature representation by IDUS (Unsupervised Learning).
3: Extract the backbone model U-Net from IDUS, and re-initialize its segmentation head.
4: Use \( \{X_{\text{label}}, Y_{\text{label}}\} \) to fine-tune the U-Net (Supervised Learning).
5: Output the model’s parameters \( \theta \) of the IDSS U-Net.

D. Implementation Details

1) Initialization: We use the idea of texton [28] to initialize the pseudo labels and a two-step texton selection [16] to reduce the computation time. We give a rough initialization to IDUS so that encourages the algorithm to converge. An initialization for IDUS contains the following steps.

1) We extract the feature maps of every image from the output of the first and third convolutional blocks in ResNet-18 [53], and the channel dimensions for two feature maps are principal component analysis (PCA) reduced to 8 and 16, respectively.
2) The feature maps from two layers are resized to \( 128 \times 128 \) pixels and then concatenated together. Therefore, the new feature maps contain the information of high- and low-level semantics.
3) We cluster each image’s features separately into \( 128 \) groups in order to find the local textons (i.e., each
image has 128 centroid features). For the dataset that has \( N \) images, we find \( N \times 128 \) local textons totally.

4) These local textons are clustered into 128 groups again to find the global textons. And, the global textons are assigned to each pixel according to the distance measured between its features and the nearest global textons.

5) The histogram of textons distribution around a pixel is computed within a 10 \times 10\) sliding window for smoothing purposes, where each histogram element represents the count of the corresponding global textons. Therefore, we get a \( 128 \times 128 \times 128 \) histogram map and resize it to \( 512 \times 512 \times 128 \) which has the same size as input images.

6) The normalized histograms with range \([0,1]\) are concatenated with the feature maps of Williams [24] (the implementations will be described in Section IV-C). We use these wavelet features because previous work [24] has shown their utility in modeling SAS imagery.

7) We use SLIC [52] to generate 100 superpixels for each image based on the generated features and pooling the pixel-level features into superpixel histograms via (1). And perform superpixel clustering via \( k \)-means to get the pseudo superpixel labels to drive the IDUS.

To clarify the relationship between the initialization and IDUS, the initialization for IDUS is performed at \( iteration 0 \) while \( iteration 1, iteration 2, \ldots, \text{etc.} \), are subsequent iterations. The only difference between \( iteration 0 \) and the subsequent iterations is that we use a set of initial features \( f_0 = \{ f_m \}_{m=1}^{N} \) generated by \textit{steps} 1–6 to replace the segmentation network softmax outputs \( \{ f_r \} \) as shown in Fig. 2. \textit{Step 7} is the same procedure as \textit{steps} 1–3 in Fig. 2.

2) Network Structure: The backbone model used by IDUS and IDSS is U-Net [20] equipped with ResNet-18 encoder that pre-trained on ImageNet [51]. Table I shows the decoder and segmentation head (pixel-level linear classifier) of U-Net. We use \{res.conv, res1, res2, res3, res4\} to denote the output of ResNet-18’s first convolutional layer and building blocks, respectively. The decoder output (conv5a) will be fed to the segmentation head (seg_head) to generate the final network outputs. The decoder and segmentation head parameters will be re-initialized right after updating pseudo-labels since we find that doing this can improve the network performance.

3) Training Configuration for IDUS: We train the IDUS with mini-batch size 15 on an NVIDIA Titan X GPU (12 GB) with the PyTorch package [54]. The loss function we use is the mean of dice loss [55] and cross-entropy loss. We use weights \( w_{\text{ent}} = 1/r_m \) and \( w_{\text{dice}} = 1/(r_m)^{1/2} \), where \( r_m \) was the proportion of \( m \)-th class label in training samples, to balance the class weights of two losses, respectively. Batch normalization layers [56] are added after each convolutional layer (except the segmentation head) to accelerate the loss convergence.

The model’s parameters are initialized by a uniform distribution following the scaling of [57] and optimized by Adam [58] algorithm with weight decay value \( 10^{-5} \). In every iteration, the learning rate of the model is started with \( 10^{-4} \) and decreased by a drop rate 0.1 every 100 epochs. The updating interval of superpixel labels and boundaries is 200 and we do this five times (five iterations) throughout training. Consequently, we train for 1000 epochs where every 200 epochs we update the superpixel labels and boundaries.

4) Training Configurations for IDSS: In IDSS, the training configuration of unsupervised learning is the same as that of IDUS. For the supervised part, the learning rate was set to \( 10^{-5} \) and decreased by 0.1 every 80 epochs. We first update the parameters of the decoder and segmentation head. After 40 epochs, we add the encoder parameters to the optimizer to jointly optimize.

IV. EXPERIMENTS AND RESULTS

In this section, we describe how we measure the performance of IDUS and demonstrates its efficacy against the comparison methods. Besides this, the extension IDSS will also be evaluated to show the benefits of semi-supervised (SS) training. Section IV-A describes the SAS dataset we used and the associated preprocessing performed. Section IV-B analyzes the convergence of IDUS. Section IV-C presents co-segmentation results. Section IV-D examines the quality of the unsupervised learned features. Section IV-E presents the run time of each method. Section IV-F presents the SS training results of IDSS when only some pixel-level labels are present. Finally, Section IV-G presents selected results of performing IDUS in a unsupervised fashion comparing its results to a SS

### Table I

| Layer Name | Layer Function | Dims. | # Filters | Input |
|------------|----------------|-------|-----------|-------|
| up1        | Upsampling     | 2x2   | N/A       | res4  |
| merge1     | Concatenate    | N/A   | N/A       | up1, res3 |
| conv1a     | Conv+BN+ReLU   | 3x3   | 256       | merge1 |
| conv1b     | Conv+BN+ReLU   | 3x3   | 256       | conv1a |
| up2        | Upsampling     | 2x2   | N/A       | conv1b |
| merge2     | Concatenate    | N/A   | N/A       | up1, res3 |
| conv2a     | Conv+BN+ReLU   | 3x3   | 128       | merge2 |
| conv2b     | Conv+BN+ReLU   | 3x3   | 128       | conv2a |
| up3        | Upsampling     | 2x2   | N/A       | conv2b |
| merge3     | Concatenate    | N/A   | N/A       | up1, res3 |
| conv3a     | Conv+BN+ReLU   | 3x3   | 64        | merge3 |
| conv3b     | Conv+BN+ReLU   | 3x3   | 64        | conv3a |
| up4        | Upsampling     | 2x2   | N/A       | conv3b |
| merge4     | Concatenate    | N/A   | N/A       | up1, res_conv |
| conv4a     | Conv+BN+ReLU   | 3x3   | 32        | conv4a |
| conv4b     | Conv+BN+ReLU   | 3x3   | 32        | conv4a |
| up5        | Upsampling     | 2x2   | N/A       | conv4b |
| conv5a     | Conv+BN+ReLU   | 3x3   | 16        | up5   |
| conv5b     | Conv+BN+ReLU   | 3x3   | 16        | conv5a |
| seg_head   | Conv+Softmax   | 3x3   | 7         | conv5b |
Fig. 3. Example SAS images with various textures: SH (blue), DS (green), BS (cyan), SG (red), RK (purple), SR (gold), LR (Gray), and unlabeled pixels (black). Textures from left to right are: (a) LR and seagrass; (b) RK and SR; (c) bright sand, dark sand, and LR; and (d) seagrass, bright sand, dark sand, and shadow.

TABLE II

| Class No. | Class Name          | Proportion |
|-----------|---------------------|------------|
| 1         | Shadow (SH)         | 0.003      |
| 2         | Dark Sand (DS)      | 0.096      |
| 3         | Bright Sand (BS)    | 0.090      |
| 4         | Seagrass (SG)       | 0.073      |
| 5         | Rock (RK)           | 0.018      |
| 6         | Small Sand Ripple (SR) | 0.033   |
| 7         | Large Sand Ripple (LR) | 0.221   |
| -         | Labeled Pixels      | 0.534      |
| -         | Unlabeled Pixels    | 0.466      |

The original SAS images are single-look complex (SLC) images made up of complex-valued pixels with a large dynamic range (this is typical of the SAS modality). All the comparison methods operate on the magnitude version of the SLC and IDUS is no different. Furthermore, each comparison method implements dynamic range compression to make the images suitable for processing of which we replicate for each respective work. For IDUS, we dynamic range compress the magnitude image by using Schlick’s rational mapping operator [59]. We set the target brightness [60] to 0.5. Additionally, we further use the OpenCV function “equalizeHist” [61] to equalize the gray-scale histogram of the tone-mapped images and normalize each to zero mean and unit standard deviation.

B. Convergence Analysis

To investigate the convergence of IDUS, we show the mean pixel accuracy (MPA) for training images as a function of IDUS iterations (recall iteration index in Algorithm 1). As iterations progress, the MPA and hence IDUS’s predicted segmentation map converges.
of similar textures tend to become homogeneous and do not have a noticeable change in the last several iterations.

C. Unsupervised Co-Segmentation

In image processing, the term co-segmentation is a special case of image segmentation which is defined as a joint pixel-level segmentation of semantically similar objects over a set of given images [63]. The objects can be a set of small rectangular image patches [24], or superpixels aligning the image textures’ boundaries well [15]. In SAS image segmentation [15], [24], co-segmentation is a popular way to differentiate seabed environment textures mainly due to the difficulty of obtaining labeled data since co-segmentation does not rely on labeled data. Therefore to show the effectiveness of IDUS, we compare performance with several comparison methods based on two co-segmentation strategies, one common to the comparison methods and one we tailored for IDUS. Since data labels are not used for co-segmentation, the entire dataset (113 sonar images) shown in Table II is used in the analysis.

1) Comparison Methods: We compare with three state-of-the-art methods [15], [24], [62] used in SAS seabed environment image segmentation. Following, we give the implementation details we use in generating the comparison methods as no source code is publicly available to evaluate. We make a best effort attempt to reproduce the methods as given in their respective sources. Our implementation is based on Python 3.7 code running on an Intel® Core® i9-7960X 2.80 GHz CPU with Linux operating system.

2) Lianantonakis et al. [62]: This method uses Haralick features [64] derived from the gray-level co-occurrence matrix (GLCM) and couples this with active contours to arrive at a binary class mapping. We extend this work to multiple classes by simply using the same feature descriptors as the original work but apply \( k \)-means++ [65] to cluster; a similar replication approach is used in [21]. We ran \( k \)-means++ with 100 random initializations and selected the run that produces the minimum within cluster sum of squares error in a manner consistent with [24].

3) Williams [24]: This method uses wavelet features along with spectral clustering to compute the segmentation map. We found that spectral clustering results in similar performance as using \( k \)-means++ so we opt to use for simplicity as we did in Lianantonakis et al. mentioned above; a similar replication approach is used in [21].

4) Zare et al. [15]: In this work, the feature sets are produced by Sobel edge descriptors (Sobel) [66], histograms of oriented gradients (HOGs) [67], and local binary pattern (LBP) features [68]. For each feature descriptor, we use the same sliding window strategy of Lianantonakis et al. [62] to derive a feature vector for each pixel.

For the comparison methods which use hand-crafted feature descriptors, we summarize the co-segmentation strategy in Fig. 6, including four main steps.

1) Feature extraction to extract a set of feature maps from the input sonar images.
2) Performing SLIC [52] on the extracted features to generate superpixel features by (1).
3) Unsupervised cluster all the superpixels and provide each superpixel a cluster assignment.
4) Pixels inside a superpixel are assigned the same cluster by (2) to get pixel-level co-segmentation.

For the co-segmentation strategy of IDUS shown in Fig. 7, we train a segmentation network by IDUS on the entire dataset first, and then directly use the network outputs computed from the input sonar images as the final co-segmentation results. The difference between the strategy for IDUS (see Fig. 7) and the strategy in Fig. 6 is that the co-segmentation of hand-crafted feature descriptor requires four steps, including feature extraction, superpixel generation, superpixel clustering, and mapping superpixel cluster assignments to each pixel. However, IDUS integrates these four steps into one single model (i.e., learning algorithm) and is performed jointly while the model is optimized.

We use the confusion matrix as a criterion to evaluate the co-segmentation performance of IDUS and the comparison methods on the entire dataset (113 images). A confusion matrix shows the proportion of an assigned class predicted by the algorithm for a given ground truth class. Ideally, the ground truth class and the predicted class overlap entirely giving a proportion of one. However, in practice, the predicted class by co-segmentation usually do not perfectly match with the ground truth class resulting in a proportion less than one.

As a result of performing unsupervised segmentation, class assignments by the algorithm must be mapped to ground-truth classes. We perform this assignment by initially using a random disjoint assignment among the possible classes and then compute the confusion matrix. Next, we re-sort the columns of the matrix in such a way so that the sum of diagonal elements in the new confusion matrix is maximized. For the co-segmentation of the comparison methods, we set the cluster number same as the number of ground-truth classes (seven class). For IDUS, we set the dimension of the network softmax output as seven. Thus, we use a confusion matrix with size $7 \times 7$ as the criterion for evaluating different methods.

Since the ground truth is semi-labeled, the unlabeled pixels do not contribute to the procedure of computing confusion matrix during testing (recall the proportion of unlabeled pixels in Table II).

Fig. 8 shows the confusion matrices of IDUS and the comparison methods. Table III reports each method’s MPA, which is derived by the mean of diagonal elements in the confusion matrix. As shown in Fig. 8 and Table III, IDUS results in superior co-segmentation results [see Fig. 8(d)] over the comparison methods [see Fig. 8(a)–(c)].

Upon examining the confusion matrix, we notice some comparison methods could not provide enough discrimination ability to differentiate certain classes well. For example, Rock (RK) and Small Sand Ripple (SR) are clustered into the same class in Williams [24] [see Fig. 8(b)]. However, such misclassification problems are diminished by IDUS [see Fig. 8(d)].

Although IDUS demonstrates the best performance in confusion matrices, the evaluation assumes each class only contains one texture type. In fact, some features might be multi-modal. In other words, one class may contain several feature types resulting in a single class being distributed among several subclasses in the latent space. For example, the category of Large Sand Ripple (LR) contains two different scales of ripple textures (in Fig. 3(a) and (b), the scales of LRs appear different). Besides this, the similarities between texture types are also different and some classes can be grouped into more general classes (e.g., Bright Sand and Dark Sand can be grouped into “Sand,” and LR and SR can be grouped into “Sand Ripple”). Therefore, the number of classes (recall we use seven classes/clusters) may be suboptimal for grouping textures in feature space, and it is necessary to evaluate the performance of the co-segmentation with different numbers of clusters. However, due to the limitations of the confusion matrix, the evaluation results are not easy to analyze.
when the co-segmentation number of clusters are different from the number of ground truth classes. To address this, we use normalized mutual information (NMI) [17] as a new metric to replace the confusion matrix for evaluating the co-segmentation results with various clusters numbers.

NMI is a metric used in DeepCluster [17] to evaluate the performance of unsupervised clustering methods. From the view of information theory, the NMI measures the information shared between the predicted classes and the ground truth classes for unsupervised segmentation. A higher NMI indicates better performance since the information overlap between the predicted classes and the ground truth classes is higher. NMI can be directly derived from a contingency matrix without the consideration of class assignment mismatch problem existing in Fig. 8. In other words, we can use a contingency matrix without column re-sorting to compute NMI.

More precisely, given a set of unsupervised predicted classes, $A$, and ground truth classes, $B$, the NMI is defined as $NMI(A; B) = \frac{(I(A; B))/(H(A)H(B))^{1/2}}$, where $I(A; B)$ is the mutual information between $A$ and $B$, $H(\cdot)$ is a measure of the entropy of classes distribution.

To compute NMI, we extract the decoder outputs from the segmentation network [the model used for the evaluation in Fig. 8(d)] of IDUS as a new set of feature maps after training and use the same co-segmentation strategy shown in Fig. 6. This results in different co-segmentation results each corresponding to different number of unsupervised predicted classes. We show co-segmentation of IDUS and comparison methods using $k$-means clustering of the learned feature space (recall Fig. 6, it is superpixel clustering) over a range of the number of clusters from 3 to 20. This range of cluster cardinality allows us to explore the texture types from coarse to fine. Recall the implementation details of Zare et al. [15], in which they use Possibilistic Fuzzy Local Information C-Means (PFLICM) for unsupervised clustering. We found PFLICM does not match the dataset used in our work [see Fig. 8(c)]. For a fair comparison, we use $k$-means instead of PFLICM [15] to cluster the superpixels of this method.

Fig. 9 shows the NMI scores of each method for a given number of clusters. The results demonstrate that IDUS performs better than the comparison methods regardless of the number of clusters. In other words, IDUS features have a more powerful feature representation for seabed environment textures. We notice that the NMIs of the Lianantonakis et al. [62] are lower in the case of small cluster numbers and higher in large cluster numbers. Such a phenomenon demonstrates that the features from this method are multi-modal for some types of textures requiring several sub-clusters to be grouped to yield good results for the final assigned classes.

D. Feature and Training Robustness

Although the co-segmentation strategy is widely used in SAS image segmentation [15], [24], it is not convenient to process new sonar images outside the training dataset: the strategy must add the new images into the existing dataset and perform the whole process again to obtain new co-segmentation results. To mitigate this issue, we can label some of the sonar images and train a linear classifier on the extracted features which are trained on the initial dataset exclusively. In this manner, as new images arrived, they are segmented...
Fig. 9. NMI for various choices of cluster numbers. The results demonstrate that IDUS performs better than competing alternatives, independent of the number of clusters.

TABLE IV
NAMES AND SAMPLE PROPORTIONS OF DIFFERENT CLASSES IN THE TRAINING SET (80 SONAR IMAGES) USED FOR TRAINING THE CLASSIFIERS OF UNSUPERVISED FEATURES

| Class No. | Class Name      | Proportion |
|-----------|-----------------|------------|
| 1         | Shadow (SH)     | 0.002      |
| 2         | Dark Sand (DS)  | 0.098      |
| 3         | Bright Sand (BS)| 0.097      |
| 4         | Seagrass (SG)   | 0.078      |
| 5         | Rock (RK)       | 0.018      |
| 6         | Small Sand Ripple (SR) | 0.038 |
| 7         | Large Sand Ripple (LR) | 0.215 |
| -         | Labeled Pixels  | 0.546      |
| -         | Unlabeled Pixels| 0.454      |

TABLE V
NAMES AND SAMPLE PROPORTIONS OF DIFFERENT CLASSES IN THE TESTING SET (33 SONAR IMAGES) USED FOR EVALUATING THE PERFORMANCE OF UNSUPERVISED FEATURE CLASSIFIERS

| Class No. | Class Name      | Proportion |
|-----------|-----------------|------------|
| 1         | Shadow (SH)     | 0.003      |
| 2         | Dark Sand (DS)  | 0.091      |
| 3         | Bright Sand (BS)| 0.072      |
| 4         | Seagrass (SG)   | 0.063      |
| 5         | Rock (RK)       | 0.019      |
| 6         | Small Sand Ripple (SR) | 0.023 |
| 7         | Large Sand Ripple (LR) | 0.236 |
| -         | Labeled Pixels  | 0.507      |
| -         | Unlabeled Pixels| 0.493      |

using this classifier. If the new images are labeled, the existing feature extractor can be utilized (which is computational expensive to regenerate) but a new linear classifier can be computed (which is computational inexpensive to regenerate).

Inspired by the experiment design of [69], we use labeled data to train a linear classifier on top of frozen feature sets to evaluate the feature quality or discriminative ability of the learned unsupervised features. If the features provide correct discrimination among ground truth classes, the linear classification is expected to yield good results. Here, our task is a pixel-level classification so we directly train a linear classifier based on every pixel feature. This is a modification from [69] which trains a linear classifier on the feature maps of images.

Our linear classifier is created by using $1 \times 1 \times D$ convolutional kernels to implement $C$ linear functions, where $C$ is the number of classes and $D$ the feature dimensions. We separate the entire dataset into a train-test split of 7:3 giving 80 images for the training set and 33 for the testing set. The class proportions for training set and testing set are shown in Tables IV and V, respectively. We first train IDUS in an unsupervised fashion using only the training set. Next, we use the training set ground truth to train a linear classifier on the decoder outputs of the frozen IDUS network. For the comparison methods (see Section IV-C), we use the feature extraction step of the co-segmentation strategy shown in Fig. 6 to extract a set of feature maps from the training set for each method. Next, we use ground truth to train a linear classifier on the feature maps using the same classifier specified above. The loss function and optimizer are the same as the training configurations of Section III-D3; however, the learning rate starts from 0.01 and decreases by 0.1 every 30 epochs. Since the ground truth is semi-labeled, the unlabeled pixels are not to be used to compute the loss function (i.e. their associated loss has a weight of zero). During the evaluation of the testing set, we extract the feature from testing sonar images (for IDUS, the feature is the decoder outputs of frozen segmentation network; for the comparison methods, the feature extraction uses a hand-crafted feature via a sliding window to extract feature) and then input them to the classifier to get the final segmentation result.

We use pixel accuracy (PA) for each class and the corresponding MPA to evaluate the performance of the linear classifiers. The equation to compute the PA $P_c$ for class $c$ is defined by $P_c = (\sum_{i=1}^{N_p} I(a_i == c, b_i == c) / \sum_{i=1}^{N_p} I(b_i == c))$, where $a_i$ and $b_i$ is the classifier prediction and ground truth of $i$th pixel, respectively, and $N_p$ is the total number of
Fig. 10. To better understand the performance in low training data scenarios, we train each method using the whole training set and then train using a random 6.25% subset. The train-test split is with a ratio of 7:3 giving 80 images for the training set (see Table IV) and 33 for the testing set (see Table V). The training subset is five images randomly selected from the training set. We report the testing set MPAs for each method trained on the training subset (five images) and the whole training set (80 images). Higher numbers indicate better performance. As shown in the figure, the benefits of IDUS over state of the art are even more pronounced when training is limited.

Table VI shows the PAs and MPA for each method on the testing set shown in Table V. Since the dataset is semi-labeled, unlabeled pixels are not considered in the evaluation. As shown in the table, IDUS has the best feature quality over all the classes except for the Shadow class. Note that the work of Zare et al. [15] has an accuracy of 1.0 for the shadow class which is a result of assigning the majority of pixels to this class.

1) Performance Versus Training Size: Deep learning-based algorithms usually work well when abundant training data is provided. However, when little labeled data are available, the performance may dramatically decline due to over-fitting. Therefore, to better understand the performance of IDUS and competing state-of-the-art methods in low training data scenarios, we randomly select five images from the training set (see Table IV) and extract the corresponding feature maps for each method. Next, we train a linear classifier on the extracted five feature maps for each method and test the linear classifier on the testing set (see Table V). Fig. 10 shows the testing set MPAs of training on five and 80 images. As shown in the figure, the large MPA gap between IDUS and the second-best method Williams [24] indicates IDUS achieves superior results even in the low training data scenario.

Although we show IDUS obtains good performance in the low training data scenario, the effectiveness may be from selection bias of the training samples. To examine this issue, we repeat the same experiment above ten times using random subsets of 6.25% (five images) of the 80 training images. We report the results as MPA in Fig. 11. It is noteworthy to mention that we use the same training subsets for the evaluation of each algorithm. As revealed by Fig. 11, IDUS achieves the highest MPA with little variation based on the exact choice of training image set.

E. Computational Burden

We evaluate the run-time of all the methods and display the results in Table VII. The run-time includes the time of feature extraction and linear classification (recall Section IV-D). The table reports the average inference time of each method in seconds per image of the test set; lower numbers indicate better performance. Since the comparison methods (Lianantonakis et al. [62], Williams [24], and Zare et al. [15]) are designed for execution on a CPU architecture, we also report the inference time of IDUS on the CPU to provide a fair comparison. As shown in Table VII, the inference speed of IDUS is much faster (even in the worse case, on CPU) than the second-best method Zare et al. [15]. The fastest speed is obtained by running IDUS on a GPU and is sufficient for real-time application.

### Table VII

| Method                | Run-Time [s/image] |
|-----------------------|--------------------|
| Lianantonakis, et al. [62] | 858                |
| Williams [24]         | 31                 |
| Zare, et al. [15]     | 4.6                |
| IDUS (CPU)            | $1.0 \times 10^{-1}$ |
| IDUS (GPU)            | $7.0 \times 10^{-3}$ |
TABLE VI
PAS OF EACH CLASS ALONG WITH THE CORRESPONDING MPA FOR EACH METHOD EVALUATED ON THE TESTING SET (33 SONAR IMAGES, SEE IN TABLE V). SINCE THE DATASET IS SEMI-LABELLED, UNLABELLED PIXELS ARE NOT EVALUATED. A HIGHER VALUE INDICATES BETTER PERFORMANCE. BEST RESULTS FOR EACH CLASS AND BEST MPA ARE IN BOLD. IDUS HAS THE HIGHEST PAS FOR MOST CLASSES AND THE BEST MPA DEMONSTRATING ITS SUPERIOR PERFORMANCE OVER COMPETING METHODS.

| Method                     | Pixel Accuracy |
|----------------------------|----------------|
|                            | SH  | DS  | BS  | SG  | RK  | SR  | LR  |
| Lianantonakis, et al. [62] | 0.00| 0.64| 0.46| 0.17| 0.67| 0.69| 0.79| 0.49|
| Williams [24]              | 0.72| 0.04| 0.84| 0.23| 0.85| 0.25| 0.97| 0.56|
| Zare, et al. [15]          | 1.00| 0.00| 0.21| 0.54| 0.62| 0.00| 0.01| 0.34|
| IDUS                      | 0.81| 0.87| 0.87| 0.69| 0.96| 0.45| 0.98| 0.80|

Fig. 12. MPA for different training set sizes; higher values indicate better performance. IDSS outperforms comparison methods, especially in the case of very limited training (five samples).

F. Comparisons Against Supervised Deep Learning Methods

Following Algorithm 2, we use IDSS to fine-tune the network on using different size training sets of labeled dataset to explore the performance improvement of combining IDUS with supervised training. The train-test splits we used are the same as mentioned in Section IV-D. In the unsupervised learning step (Step 2 in Algorithm 2), we use 80 sonar images (whole training set) without ground truths \( \{X_{\text{unlabel}}\} \) to learn features by IDUS. In the supervised learning step (Step 4 in Algorithm 2), we randomly pick a subset of labeled images \( \{X_{\text{label}}, Y_{\text{label}}\} \) from the training set to fine-tune the U-Net.

Fig. 12 shows the plot of MPAs on different training set sizes of labeled images \( \{5, 10, 15, 20, 25\} \). We also train a state-of-the-art U-Net [20] and the recent supervised approach of Rahnemoonfar et al. [29] on different training set sizes to compare the performance of best-known supervised deep networks to IDSS.\(^3\) For fairness in comparison, when training U-Net, Rahnemoonfar et al. [29] and IDSS, the training image sets, numerical optimizer, learning rate strategy and batch size are all the same as the training configurations shown in Section III-D3. To be clear, U-Net and Rahnemoonfar et al. are trained with known ground truth pixel labels. However, since a given training image is only partially-labeled, the unlabeled pixels (recall in Table IV) are not considered in the loss function computation when training the comparison models.

We do not report the accuracy of training set sizes larger than 25 since the accuracy does not improve much beyond this size. Fig. 12 results once again demonstrates that IDSS benefits are even more pronounced when training data is not abundantly available. Note especially that in the lowest training data scenario of five images, IDSS provides the largest improvement (0.128 MPA) over the second best method. Note also that Rahnemoonfar et al. [29] do not perform particularly well on our SAS dataset. We conjecture this for a couple of reasons: 1) their approach was designed for sidescan sonar and 2) being a purely supervised method, the method in [29] is highly reliant on generous training, which is practically infeasible and not available in our SAS dataset.

G. Visualization of Select Results

To visualize network predictions, we use the same train-test splits from Section IV-D to train a supervised U-Net as in Section IV-F and IDUS. We use a subset of 15 images of the training set to train IDSS.

The prediction results of IDUS, IDSS, and the U-Net on the test set are shown in Fig. 13. In the top row of Fig. 13, although IDUS is trained in an unsupervised fashion, the predictions of
IDUS, SS IDSS, and supervised U-Net are almost the same in the labeled region, especially for the LR. In the bottom row of Fig. 13, IDUS and IDSS provide more consistent predictions for the unlabeled regions and exhibit homogeneity in labeling similar textures. We see that the U-Net’s predictions do not have these nice properties especially in unlabeled regions. We posit a possible reason for this: the U-Net overfits since the data is partially and sparsely-labeled. IDUS and IDSS exploit custom feature initialization to group local cluster textures having similar spatial characteristics into a single class assignment thereby mitigating overfitting. Fig. 13 shows IDUS and IDSS segment SAS image regions more agreeably with the ground truth than the U-Net.

V. CONCLUSION

We propose an unsupervised method for SAS seabed image segmentation which works by iteratively clustering on superpixels while training a deep model. Our proposed method, Iterative Deep and Unsupervised Segmentation (IDUS), does not require large amounts of labeled training data which is difficult to obtain for SAS. We demonstrate that IDUS obtains state-of-the-art performance on a real-world SAS dataset and show performance benefits by comparing against existing methods. Additionally, we develop an extension of IDUS called IDSS that is able to work in a weakly or SS manner. IDSS is also evaluated against supervised state-of-the-art methods and shown to yield superior results. Finally, we show inference (run) times for IDUS/IDSS which indicate that it can easily be deployed for real-time processing onboard low size, weight, and power (SWaP) systems.

ACKNOWLEDGMENT

The authors thank Dr. J. Tory Cobb of the Naval Surface Warfare Center, Panama City, FL, USA, for providing the data used in this work.

REFERENCES

[1] P. T. Gough, “A synthetic aperture sonar system capable of operating at high speed and in turbulent media,” *IEEE J. Ocean. Eng.*, vol. JOE-11, no. 2, pp. 333–339, Apr. 1986.
[2] M. P. Hayes and P. T. Gough, “Results from an experimental synthetic aperture sonar,” in *Acoustical Imaging*, Boston, MA, USA: Springer, 1991, pp. 455–466.
[3] M. P. Hayes and P. T. Gough, “Broad-band synthetic aperture sonar,” *IEEE J. Ocean. Eng.*, vol. 17, no. 1, pp. 80–94, Jan. 1992.
[4] H. J. Callow, “Signal processing for synthetic aperture sonar image enhancement,” Ph.D. dissertation, Dept. Elect. Electron. Eng., Univ. Canterbury, Christchurch, New Zealand, 2003.
[5] S. A. Fortune, M. P. Hayes, and P. T. Gough, “Statistical autofocus of synthetic aperture sonar images using image contrast optimisation,” in *Proc. MTS/IEEE Oceans. Ocean Odyssey. Conf.*, Nov. 2001, pp. 163–169.
[6] A. J. Hunter, M. P. Hayes, and P. T. Gough, “Simulation of multiple-receiver, broadband interferometric SAS imagery,” in *Proc. OCEANS*, vol. 5, Sep. 2003, pp. 2629–2634.
[7] D. Brown, D. Cook, and J. Fernandez, “Results from a small synthetic aperture sonar,” in *Proc. OCEANS*, 2006, pp. 1–6.
[8] C. B. DuVal, A. C. Trembanis, and A. Skarke, “Characterizing and hindcasting ripple bedform dynamics: Field test of non-equilibrium models utilizing a fingerprint algorithm,” *Continental Shelf Res.*, vol. 116, pp. 103–115, Mar. 2016.
[9] A. Trembanis, C. DuVal, J. Beaudoin, V. Schmidt, D. Miller, and L. Mayer, “A detailed seabed signature from hurricane sandy revealed in bedforms and scour,” *Geochim., Geophys., Geosyst.*, vol. 14, no. 10, pp. 4334–4340, Oct. 2013.
[10] A. C. D. Davis, R. G. Kvitk, C. B. A. Mueller, M. A. Young, C. D. Storlazzi, and E. L. Phillips, “Distribution and abundance of rippled scour depressions along the California coast,” *Continental Shelf Res.*, vol. 69, pp. 88–100, Oct. 2013.
[11] A. Denny, R. Hansen, T. Saebø, and R. Pedersen, “The use of synthetic aperture sonar to survey seafloor massive sulfide deposits,” *J. Ocean Technol.*, vol. 10, pp. 36–53, Apr. 2015.
[12] V. Yordanova, B. Gips, T. Pufarfo, and S. Dugelay, “Coverage path planning for mine countermeasures: Adapting track orientation,” in *Proc. OCEANS-Marseille*, Jun. 2019, pp. 1–7.
[65] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” in Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms, 2007, pp. 1027–1035.

[66] H. Frigui and P. Gader, “Detection and discrimination of land mines in ground-penetrating radar based on edge histogram descriptors and a possibilistic K-nearest neighbor classifier,” IEEE Trans. Fuzzy Syst., vol. 17, no. 1, pp. 185–199, Jun. 2009.

[67] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., Jun. 2005, vol. 1, no. 1, pp. 886–893.

[68] Z. Guo, L. Zhang, and D. Zhang, “A completed modeling of local binary pattern operator for texture classification,” IEEE Trans. Image Process., vol. 19, no. 6, pp. 1657–1663, Jun. 2010.

[69] R. Zhang, P. Isola, and A. A. Efros, “Split-brain autoencoders: Unsupervised learning by cross-channel prediction,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 1058–1067.

Yung-Chen Sun received the B.S. degree from the Sun Yat-sen University, Guangzhou, China, in 2018, and the M.S. degree (Hons.) from The Hong Kong Polytechnic University, Hong Kong, in 2019. He is currently pursuing the M.S. degree with the Electrical Engineering Department, The Pennsylvania State University, University Park, PA, USA. He is also a member of the Information Processing and Algorithms Laboratory, The Pennsylvania State University. His research interests include deep learning, semantic segmentation, and unsupervised learning.

Isaac D. Gerg received the B.S. degree (Hons.) in computer engineering and the M.S. degree in electrical engineering from The Pennsylvania State University (PSU), University Park, PA, USA, in 2004 and 2008, respectively, where he is currently pursuing the Ph.D. degree in electrical engineering with the Information Processing and Algorithms Laboratory. He is also with the Information Processing and Algorithms Laboratory, PSU. He is also with The Pennsylvania State University Applied Research Laboratory (PSU-ARL), State College, PA, where his research interests include remote sensing and machine learning. Mr. Gerg was a co-recipient of the PSU-ARL Engineering Award of Excellence for his work on synthetic aperture sonar beamforming.

Vishal Monga (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA, in August 2005. From October 2005 to July 2009, he was an Imaging Scientist with Xerox Research Labs, Webster, NY, USA. He has been with the Faculty of Electrical Engineering and Computer Science, The Pennsylvania State University, University Park, PA, USA, since Fall 2009. He has also been a Visiting Researcher with Microsoft Research, Redmond, WA, USA; and a Visiting Faculty with the University of Rochester, Rochester, NY. His group’s research focuses on optimization-based methods with applications in signal and image processing, and learning and computer vision. Dr. Monga was an elected member of the IEEE Image Video and Multidimensional Signal Processing (IVMSP) Technical Committee from 2017 to 2019. He was a recipient of the U.S. National Science Foundation CAREER Award and the 2016 Ruth and Joel Spira Teaching Excellence Award. He has served as an Associate Editor for the IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, the IEEE SIGNAL PROCESSING LETTERS, and the Journal of Electronic Imaging (SPIE). He is currently serving on the IEEE Bio Imaging and Signal Processing (BISP) Technical Committee and is a Senior Area Editor of the IEEE SIGNAL PROCESSING LETTERS.