A. Data and Models Release

All training data used in this paper is available at https://github.com/mvaldenegro/marine-debris-fls-datasets/ and models and source code is available at https://github.com/agrija9/ssl-sonar-images.

B. RotNet Model Selection

In this section we describe the neural network architectures that we used for evaluating RotNet.

Similar to the work in [40], we have implemented the several CNN architectures to evaluate the performance of RotNet in sonar images. All the described network architectures are lightweight versions that try to address memory and performance challenges in underwater robotics:

ResNet [10]: Uses residual connections to improve gradient propagation through the network, this feature allows to design much deeper networks. For our experiments we used a compact variant (ResNet20) which has 20 residual layers.

MobileNet [13]: It is designed to be lightweight in order to do fast inference in mobile devices. It features depthwise separable convolutions to reduce computations (this induces a trade-off between accuracy performance and computation performance).

DenseNet [14]: It reuses features heavily. It is composed by a set of dense convolutional blocks, where each of them is a series of convolutional layers, and each layer takes as inputs the feature maps from the previous convolutional layers in the same block. For our experiments, we used DenseNet121, which is the shallowest variation as described in [14].

SqueezeNet [15]: It reduces the number of parameters while maintaining competitive performance, this is achieved by designing so called Fire modules which contain two sub-modules, one squeezes information through a bottleneck and another one expands the amount of information. According to [15], SqueezeNet has similar performance to AlexNet on ImageNet but with overall 50x less parameters.

MiniXception: This network is a modification to the Xception network [4] since it reduces the amount of computation needed (e.g. for facial emotion recognition [1]). Xception is a model that combines ideas from MobileNets by using depthwise separbale convolutions.

C. Detailed RotNet Transfer Learning Results

In this section we show additional detailed transfer learning results with RotNet, presented in Table 6.

Test Set Accuracy					
RotNet Baseline	**Layer**	**10 Samples**	**40 Samples**	**110 Samples**	**200 Samples**
ResNet20 Self-Supervised	flatten	68.22 ± 1.83%	84.40 ± 1.74%	90.06 ± 1.08%	92.07 ± 0.58%
	activation 18	66.64 ± 2.95%	85.70 ± 0.84%	93.48 ± 0.67%	96.46 ± 0.54%
	activation 17	66.19 ± 2.14%	85.28 ± 1.31%	93.62 ± 0.83%	**96.62 ± 0.56%**
ResNet20 Supervised	flatten	76.51 ± 3.30%	90.20 ± 1.23%	94.62 ± 0.75%	96.17 ± 0.46%
	activation 18	70.14 ± 4.40%	88.06 ± 1.40%	95.27 ± 0.74%	**97.27 ± 0.60%**
	activation 17	71.61 ± 3.10%	88.43 ± 1.44%	95.25 ± 0.87%	97.14 ± 0.79%
MobileNet Self-Supervised	conv_pw_11_relu	58.21 ± 1.65%	74.66 ± 1.13%	83.86 ± 1.03%	88.14 ± 0.76%
	flatten	51.08 ± 2.07%	68.32 ± 1.28%	76.68 ± 0.99%	82.35 ± 0.99%
	conv_pw_12_relu	54.97 ± 2.03%	70.63 ± 1.23%	79.30 ± 1.24%	83.31 ± 0.77%
MobileNet Supervised	conv_pw_11_relu	63.90 ± 1.64%	76.38 ± 1.72%	84.09 ± 1.27%	87.34 ± 0.84%
	flatten	60.04 ± 3.15%	72.10 ± 1.10%	78.80 ± 0.86%	81.64 ± 1.16%
	conv_pw_12_relu	59.98 ± 1.89%	74.35 ± 1.35%	80.40 ± 1.27%	83.69 ± 0.48%
DenseNet121 Self-Supervised	conv5_block15_0_relu	67.95 ± 2.17%	85.59 ± 1.20%	92.10 ± 0.90%	95.02 ± 0.43%
	conv5_block16_0_relu	68.85 ± 2.88%	86.89 ± 1.13%	92.58 ± 0.73%	94.66 ± 0.56%
	avg pool	64.51 ± 3.17%	80.96 ± 1.11%	88.26 ± 1.05%	90.17 ± 0.69%
DenseNet121Supervised	conv5_block15_0_relu	69.95 ± 3.29%	86.69 ± 0.94%	93.75 ± 0.64%	95.70 ± 0.41%
	conv5_block16_0_relu	68.97 ± 3.34%	86.26 ± 1.38%	93.56 ± 0.79%	95.61 ± 0.55%
	avg pool	63.65 ± 2.70%	81.82 ± 1.39%	88.60 ± 0.85%	91.96 ± 0.37%
SqueezeNet Self-Supervised	batch_norm_8	62.54 ± 2.40%	82.10 ± 0.88%	89.56 ± 0.85%	92.37 ± 0.58%
	batch_norm_9	39.70 ± 2.99%	55.14 ± 1.62%	61.64 ± 1.53%	64.65 ± 1.33%
	global_average_pooling2d	23.92 ± 2.63%	25.13 ± 0.98%	26.27 ± 0.80%	25.95 ± 0.98%
SqueezeNet Supervised	batch_norm_8	71.76 ± 2.61%	87.44 ± 1.40%	93.67 ± 0.92%	96.31 ± 0.40%
	batch_norm_9	56.07 ± 2.52%	73.59 ± 0.96%	80.34 ± 1.53%	83.07 ± 0.84%
	global_average_pooling2d	38.77 ± 2.61%	45.34 ± 1.59%	47.22 ± 1.23%	48.20 ± 1.15%
Minixception Self-Supervised	add_3	68.63 ± 2.13%	82.94 ± 1.41%	90.12 ± 0.73%	92.29 ± 0.70%
	add_2	66.71 ± 2.28%	85.78 ± 1.21%	92.80 ± 1.01%	95.30 ± 0.51%
	conv2d_6	52.74 ± 2.43%	65.75 ± 1.48%	70.91 ± 1.01%	73.73 ± 0.95%
Minixception Supervised	add_3	71.33 ± 3.39%	88.38 ± 0.97%	94.34 ± 0.83%	96.86 ± 0.46%
	add_2	70.34 ± 2.23%	87.98 ± 0.96%	94.63 ± 0.77%	96.91 ± 0.68%
	conv2d_6	64.24 ± 3.36%	79.87 ± 1.20%	86.68 ± 1.17%	89.39 ± 0.56%
Linear SVM	NA	63.55 ± 3.66%	85.39 ± 1.07%	92.51 ± 0.79%	95.67 ± 0.90%

Table 6. RotNet transfer learning classification accuracies on the Turntable dataset.

D. Denoising Autoencoder Model Selection

In this section we describe the neural network architecture that we used as a Denoising Autoencoder.

We selected the following encoder-decoder CNN architecture for the DAE, which based on the one reported in [40]: The encoder architecture consists of Conv2D(32, 3×3) - MaxPool(2×2) - Conv2D(16, 3×3) - MaxPool(2×2) - Conv2D(8, 3×3) - MaxPool(2×2) - Flatten() - Fully Connected(c). The decoder architecture is composed by Fully Connected($c \times n_w \times n_h$) - Reshape() - Conv2D(32, 3×3) - UpSample(2×2) - Conv2D(16, 3×3) - UpSample(2×2) - Conv2D(8, 3×3) - UpSample(2×2) - Conv2D(1, 3×3).

E. Denoising Autencoder Transfer Learning Results

In this section we show additional detailed transfer learning results with a Denoising Autoencoder, presented in Table 7.

Test Set Accuracy						
Model	**Code Size**	**Gaussian Noise (σ)**	**10 Samples**	**40 Samples**	**110 Samples**	**200 Samples**
Denoising AE	32	0.100	64.40 ± 2.98%	80.01 ± 1.72%	82.68 ± 1.20%	84.49 ± 0.64%
		0.125	62.29 ± 1.55%	75.42 ± 1.02%	78.37 ± 0.84%	79.48 ± 1.00%
		0.150	60.89 ± 2.62%	68.91 ± 1.82%	71.73 ± 0.75%	72.55 ± 1.18%
		0.175	63.03 ± 3.32%	77.28 ± 1.11%	81.20 ± 1.21%	82.62 ± 0.85%
		0.200	61.59 ± 2.35%	74.21 ± 2.13%	79.16 ± 1.17%	80.40 ± 1.15%
	64	0.100	64.97 ± 1.80%	83.34 ± 1.47%	88.06 ± 0.81%	89.65 ± 0.71%
		0.125	61.81 ± 2.59%	78.06 ± 1.24%	82.71 ± 1.29%	85.11 ± 1.0%
		0.150	67.90 ± 2.80%	81.44 ± 1.72%	86.79 ± 0.91%	88.40 ± 0.66%
		0.175	63.93 ± 3.15%	79.70 ± 1.95%	85.84 ± 0.80%	87.45 ± 0.77%
		0.200	62.18 ± 1.92%	79.84 + - 1.69%	85.05 ± 1.23%	86.69 ± 0.63%
	128	0.100	68.31 ± 1.60%	85.28 ± 1.32%	91.14 ± 1.09%	94.31 ± 0.63%
		0.125	65.05 ± 2.98%	83.42 ± 1.61%	88.94 ± 1.22%	90.07 ± 0.90%
		0.150	69.39 ± 2.24%	85.38 ± 1.60%	92.40 ± 0.87%	**94.49 ± 0.62%**
		0.175	69.81 ± 2.31%	86.82 ± 0.97%	92.37 ± 0.78%	93.96 ± 0.73%
		0.200	66.88 ± 2.98%	84.51 ± 1.52%	91.48 ± 0.60%	93.73 ± 0.94%
AE	32	-	67.53 ± 2.03%	81.56 ± 1.55%	86.54 ± 1.13%	88.31 ± 0.83%
	64	-	67.10 ± 3.01%	83.84 ± 1.54%	89.16 ± 0.92%	91.39 ± 0.53%
	128	-	67.86 ± 1.53%	83.25 ± 1.59%	90.54 ± 1.30%	**92.21 ± 0.59%**

Table 7. Denoising Autoencoder transfer learning classification accuracies on the Turntable dataset.

F. Jigsaw Model Selection

In this section we describe the neural network architecture that we used for Jigsaw self-supervised learning.

Extensive experiments were carried out to optimize the CNN feature extractor design that works as a baseline for Jigsaw. The main parameters that we varied were the number of layers, number of filters and downsampling in the feature extractor. The original Jigsaw architecture [30] uses a set of $\{64, 128, 256, 386\}$ 2DConvs. In our case, we have reduced considerably the number these filters (and hence number of parameters) to a set of $\{32, 16, 8\}$ 2DConvs, this is because our Jigsaw model is trained on the smaller Watertank dataset and such large models might lead easily to overfitting.

We used a Time Distributed Layer[1] (TDL) from Keras in order to feed image patches simultaneously through the sequential CNN feature extractor and a final decision network (classification layer). The feature extractor is composed of the following layers: Conv2D(32, 3×3) - BatchNorm() - MaxPool(2×2) - Dropout() - Conv2D(16, 3×3) - BatchNorm() - MaxPool(2×2) - Dropout() - Conv2D(8, 3×3) -BatchNorm() - MaxPool(2×2)- Dropout() - Flatten(). The TDL takes this sequential model and flattens the output predictions from every image tile to then process them through the decision network. The decision network is composed by: TimeDistributedLayer(9, None) - Flatten() - FullyConnected() - BatchNorm() - FullyConnected() - BatchNorm() - Dropout() - FullyConnected().

G. Jigsaw Transfer Learning Results

In this section we show additional detailed transfer learning results with Jigsaw, presented in Table 8.

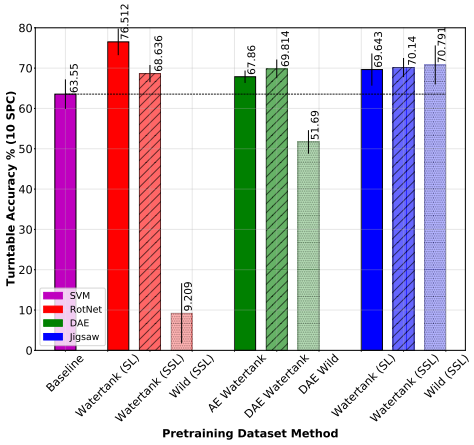
Test Set Accuracy					
Permutations	**Layer**	**10 Samples**	**40 Samples**	**110 Samples**	**200 Samples**
5	dropout_0	60.61 ± 1.68%	83.93 ± 1.25%	91.59 ± 0.63%	95.19 ± 1.09%
	dropout_1	66.41 ± 2.67%	86.17 ± 1.06%	94.51 ± 0.95%	96.74 ± 0.66%
	dropout_2	70.14 ± 2.36%	86.76 ± 2.10%	94.35 ± 1.16%	96.24 ± 0.47%
10	dropout_0	64.248 ± 2.22%	84.71 ± 1.79%	92.65 ± 1.44%	95.62 ± 0.74%
	dropout_1	64.62 ± 1.75%	86.29 ± 1.85%	94.76 ± 0.74%	**96.83 ± 0.47 %**
	dropout_2	66.20 ± 1.51%	88.55 ± 0.78%	93.95 ± 1.41%	96.77 ± 0.41%
15	dropout_0	64.0 ± 3.88%	84.40 ± 0.98%	91.84 ± 1.13%	95.16 ± 0.50%
	dropout_1	67.10 ± 1.68%	86.73 ± 1.66%	94.51 ± 1.07%	96.65 ± 0.46%
	dropout_2	68.18 ± 4.74%	85.73 ± 2.09%	93.79 ± 0.68%	95.93 ± 0.58%
20	dropout_0	60.71 ± 2.55%	83.59 ± 0.73%	91.84 ± 0.77%	95.19 ± 0.39%
	dropout_1	63.16 ± 1.89%	85.36 ± 1.51%	93.55 ± 0.63%	95.62 ± 0.87%
	dropout_2	67.25 ± 2.57%	87.16 ± 1.34%	93.36 ± 0.72%	95.59 ± 0.52%
Supervised	dropout_0	65.45 ± 2.96%	87.59 ± 1.51%	93.82 ± 1.05%	96.58 ± 0.59%
	dropout_1	69.64 ± 3.99%	89.33 ± 1.09%	94.91 ± 0.85%	**97.08 ± 0.22%**
	dropout_2	66.94 ± 1.26%	87.25 ± 1.46%	93.89 ± 1.51%	96.09 ± 0.51%

Table 8. Jigsaw transfer learning classification accuracies on the Turntable dataset.

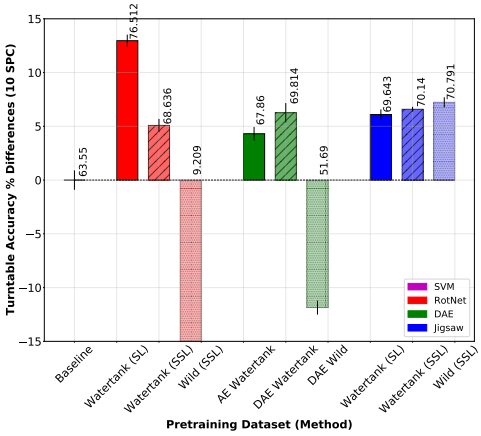
[1] https://keras.io/api/layers/recurrent_layers/time_distributed/

H. Comparison of Best Performing Transfer Learning Models

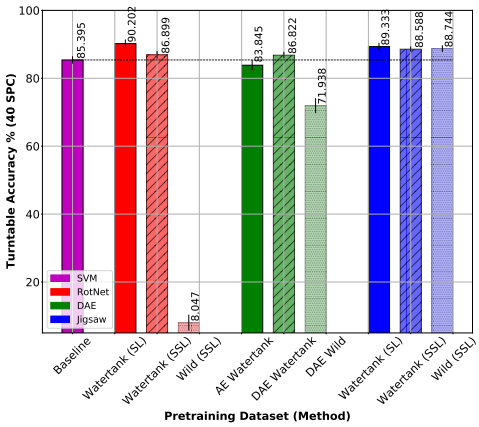
In this section we show additional comparisons of the best transfer learning results, presented in Figure 11 for selected values of samples per class (SPC).



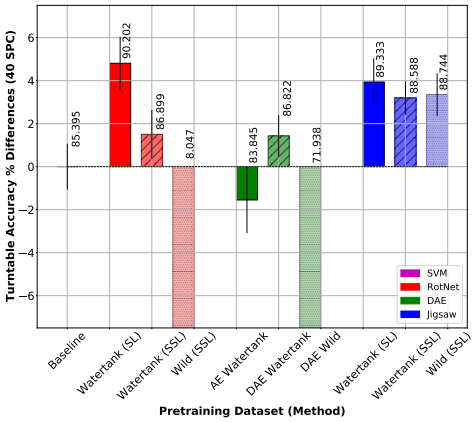
(a) 10 spc.



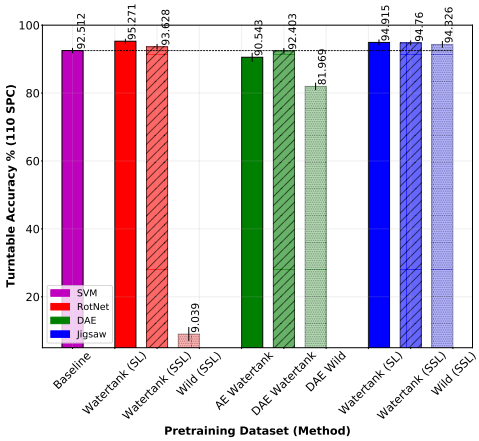
(b) 10 spc with baseline substracted.



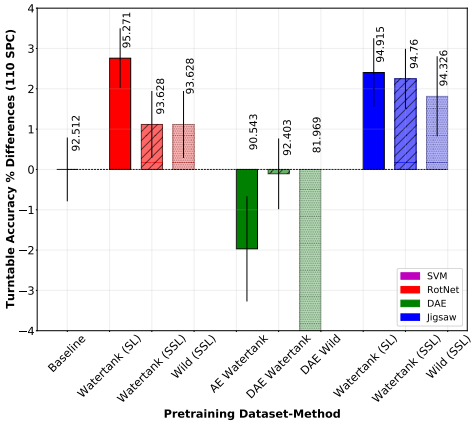
(c) 40 spc



(d) 40 spc with baseline substracted.



(e) 110 spc



(f) 110 spc with baseline substracted.

Figure 11. Comparison of the best performing SSL models pretrained on the Watertank sonar dataset (with supervision and self-supervision) and the Wild sonar dataset. 10, 40 and 110 spc cases.