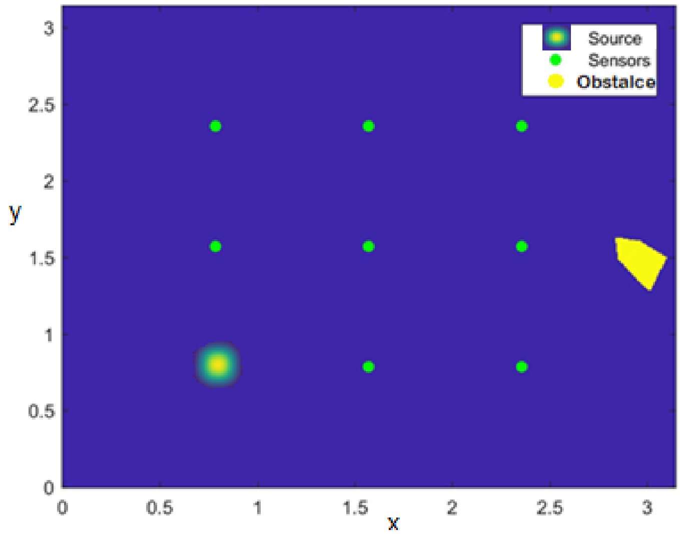


Figure 3: An illustration of the setup we use for most of the numerical experiments throughout this work. The source is a compactly supported thin Gaussian. The sensors are grid 8 grid points. An arbitrary polygon is shown here as an example.

the same FDCD approach. This FDCD scheme is an explicit compact second order accurate scheme in both space and time. The time step is limited by the Courant-Friedrichs-Lewy condition $\Delta t \leq \frac{1}{c\sqrt{2\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right)}}$. We use this to forward propagate the waves through N_{steps} time steps, and "record" the pressures at the sensors, resulting in a matrix of size $N_{steps} \times N_{sensors}$, containing the synthetically generated acoustic pressure measurements.

2.3 Formulation as a data-driven problem

We formulate the obstacle identification as a data-driven problem. Following [11], we use image segmentation techniques to find and identify the obstacles. We create binary images where pixels with the value 1 are inside the obstacle, and pixels with the value 0 are in the background. To randomly generate the obstacles, we create arbitrary polygons by generating a random number of edges, random edge lengths, and random angles between the edges. The number of edges and edge lengths were generated using a normal distribution while the angles were generated using a uniform distribution. We created a total of $N_{samples}$ arbitrary polygons, for the obstacles data which we denote as $\{\mathcal{O}_m\}_{m=1}^{N_{samples}}$. We also call these labels.

We use an initial source of a compactly supported thin Gaussian. An illustration of the setup including the source, the locations of the sensors and an arbitrary polygonal obstacle is given in Figure 3. Using FDCD to compute the forward propagation problem, we compute the data in the sensors. The obstacle is set in the domain by setting the wave propagation velocity in the domain to $c^2(1-\mathcal{O}_m)$ (for each $m = 1, ..., N_{samples}$). Inside the obstacle the wave propagation speed is set to 0 (no propagation), and outside it the speed is c^2. Then, we generate another arbitrary polygon and solve the forward problem again with a different obstacle image. This is done for all $N_{samples}$. We eventually achieve a three dimensional matrix of size $N_{samples} \times N_{steps} \times N_{sensors}$, with corresponding binary images of obstacles used to simulate the forward process, a three dimensional matrix of size $N_{samples} \times N_x \times N_y$. The data-driven problem is then to fit a set of sensor recordings $\{u_m(t_n, x_k, y_k)\}_{n=1,k=1}^{N_{steps},N_{sensors}}$ for each sample $m = 1, ..., N_{samples}$, to a corresponding binary image of the obstacle $\{\mathcal{O}_m\}_{m=1}^{N_{samples}}$.

3 Deep learning framework

We propose using a method composed of three steps:

1. Preprocessing and feature extraction using TR.

2. Convolutional layers that predict the location and shape of the obstacle.

3. Physics-informed loss term used only during training.

The input to the method is the sensor recordings. We first apply TR to get the reconstruction of the source. Then, we train a convolutional neural network that learns to extract the obstacle from the image. Last, we introduce a physics-informed loss term that utilizes the wave equation, to create another loss term for training the network. The additional loss term acts as a penalty term, yielding better training and overall better performance of the method. We now discuss each step and the contribution to the overall method.

3.1 TR based feature extraction

In this step we use FDCD, as discussed in section 2.2, to backward propagate the waves using the sensor recordings. In the backward step we do not have any information about the obstacle. To use the FDCD, we first initialize the grid with 0 values. We place the values recorded in the sensors in the corresponding coordinates of the grid at the final time step ($u_{i,j}^{N_{steps}}$), and at the time step before it ($u_{i,j}^{N_{steps}-1}$). We then compute $u_{i,j}^{N_{steps}-2}$ and substitute the computed values at the sensors coordinates with the recorded values there from the forward step, at that time. We do this iteratively using (2) until we get $u_{i,j}^{0}$, which is the desired reconstruction.

Hence, this step transforms each sample from size $N_{steps} \times N_{sensors}$ to a matrix of size $N_x \times N_y$. Note that in relevant applications $N_{sensors} \ll N_{steps}$, while $N_x \approx N_y$, so we start with an uneven rectangular-like input and get a square-like output. One advantage of this transformation is that we immediately get the desired image shape $N_x \times N_y$ which allows us to use encoder-decoder (image to image) architectures. We achieve this in a mathematically meaningful way, without using any upsampling or interpolation methods.

Furthermore, this process performs a type of feature extraction. As seen in figure 4, its result is a sparse-like image with higher values concentrated around the source, and smaller values approaching zero farther away from the source. By switching to a different representation of the data we enable the network to learn from new rich features that could improve the performance of the model.

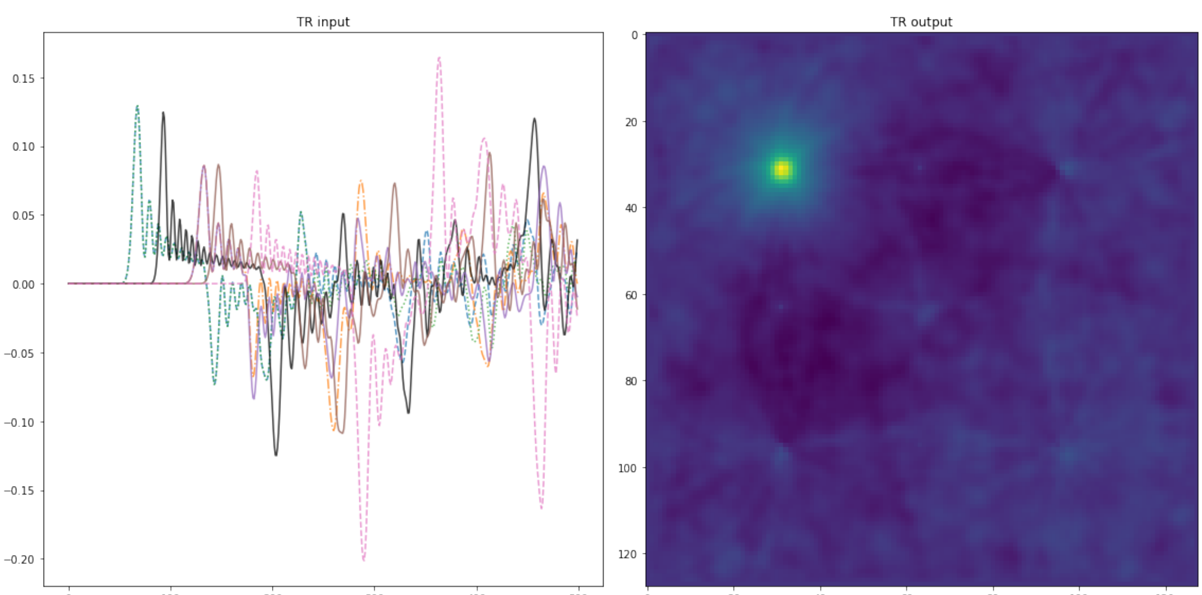


Figure 4: An example of the TR step. On the left-hand side we have one sample taken from the raw dataset. It is composed of multiple sensor recordings over time. On the right-hand side we can see the output of the TR preprocessing step, which is a source reconstruction in the form of a single image.

3.2 Convolutional architecture

This step involves a deep neural network, trained to predict the obstacle images. During training, we use Stochastic Gradient Descent (SGD) to find the best set of parameters of the network that minimizes a predefined loss function. We use the Dice loss function [16, 17] defined by:

$$Dice := \frac{1}{N_{samples}N_xN_y}\sum_{m=1}^{N_{samples}}\frac{2\langle\mathcal{O}_m,\tilde{\mathcal{O}}_m\rangle_F}{\|\mathcal{O}_m\|_F^2+\|\tilde{\mathcal{O}}_m\|_F^2}, \tag{3}$$

where $\{\mathcal{O}_m\}_{m=1}^{N_{samples}}$, $\{\tilde{\mathcal{O}}_m\}_{m=1}^{N_{samples}}$ are respectively the sets of true and predicted labels, and $\langle\cdot,\cdot\rangle_F$, $\|\cdot\|_F^2$ are the Frobenius norm. Note that the predicted labels $\{\tilde{\mathcal{O}}_m\}_{m=1}^{N_{samples}}$ are not binary - they are probability images. For numerical stability, a small $0<\epsilon$ value is often added to the numerator and denominator. The Dice loss is a differentiable metric for the comparison between the true and predicted images.

To extract the obstacle from the reconstruction of the initial condition made by TR in the previous step, we employ a variant of the DeepLabv3+ [18] architecture. This architecture uses an encoder-decoder structure, where the input and output of the neural network have the same shape. It was designed for the semantic segmentation problem from computer vision, which is the task of assigning pixel-level labels for an image. DeepLabv3+ has been shown to achieve state-of-the-art (SOTA) results on benchmark semantic segmentation datasets such as PASCAL VOC 2012 [19]. Due to the similarity of the obstacle problem to semantic segmentation, using a SOTA architecture from this domain is a natural choice.

DeepLabv3+ requires a backbone neural network to form the basis of the architecture. In the original paper, the authors used the Xception [20] architecture. In this paper we use a lighter version of Xception. We reduce the number of convolutional filters in each layer of the middle flow part of the original architecture by a factor of 4. The full modified DeepLabv3+ architecture ends up having around four million trainable parameters, which is a relatively small amount in terms of modern deep learning.

3.3 Physics-informed loss term

In addition to the Dice loss defined above, we add another loss term to the training procedure. This term is based on the FDCD described in 2, and is unique to the wave problem, since FDCD is a numerical solver of the wave equation. This makes the network aware of the wave equation, and the method is uniquely tailored for the wave problem data. This method can be extended to other problems as well, using an appropriate solver of the desired problem.

During the training iterations, every sample goes through the network to produce a prediction, which is a predicted image of the obstacle. We then use the forward FDCD for the predicted image (exactly how it was used to create the data in section 2.3), and compute the sensor recordings from the forward process. Unlike the true obstacle image, the predicted one is not binary, but is a probability image. The method is appropriate, since the term $c^2(1-\tilde{\mathcal{O}})$ produces velocities close to 0 inside the obstacle and close to c^2 elsewhere (assuming the network trains correctly). Therefore, we produce comparable sensor recordings (true versus predicted). We compare the sensor recordings to the true sensor recordings, which are the inputs to the TR block. We then compare the true sensor recordings to the predicted ones using the Mean Squared Error (MSE) defined by

$$PI = \frac{1}{N_{samples}N_{steps}N_{sensors}}\sum_{m=1}^{N_{samples}}\sum_{n=1}^{N_x}\sum_{k=1}^{N_y}\left|u_m(t_n,x_k,y_k)-\tilde{u}_m(t_n,x_k,y_k)\right|^2. \tag{4}$$

If the network trained perfectly (the prediction and the truth are identical), the same forward solver would have been applied to the same obstacle image, producing the same sensor recordings and the physics-informed loss would be 0. However, this is an impossible scenario. The better the network learns, the lower the physics-informed loss. Since the FDCD solver used to compute the forward process is linear (operates by summations and multiplications only), the automatic differentiation mechanism in Tensorflow 2 [21] is able to compute the gradients and no additional implementation is needed.

To train, we take a combination of the Dice loss and the physics-informed loss. We train the network and print the values of the Dice loss and the physics informed loss. We found that the two losses are of the same order of magnitude. Thus, the loss we use for the training is the sum of the two losses. We recommend checking this before training, because if, for a different setup, one discovers large differences between the values, then one may consider weighting the two terms of the loss, giving a larger weight to one of them to bridge the gap. Failing to do so may make the smaller loss term redundant and insignificant, leading to saturation (the network would not converge).

4 Numerical tests and results

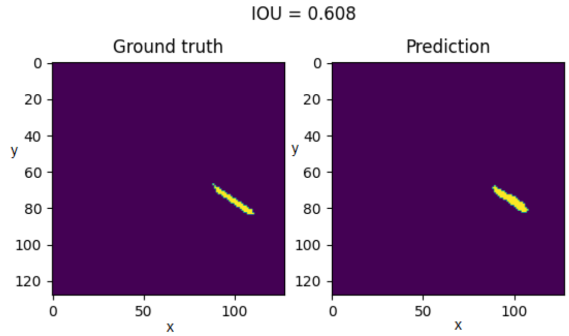
We created 60,000 samples as described in section 2.3. We chose a grid of size 128×128 with a physical size $[0, \pi] \times [0, \pi]$. We used 8 fixed sensors as shown in Figure 3. We emphasize that randomly generating the obstacles, as described in section 2.3, creates an enormous number of possible samples, and we are using only 60,000. The total storage size of the dataset is 8.1GB. The parameters we used are $c = 1484 \frac{m}{s}$ (average acoustic wave propagation speed in the Mediterranean sea), $\Delta x = \Delta y = \frac{\pi}{127} \approx 0.025$, and $\Delta t = \frac{1}{c\sqrt{2\left(\frac{1}{\Delta x^2} \frac{1}{\Delta y^2}\right)}} \approx 8.33 \cdot 10^{-6}$. The total number of time steps is 500, so that the wave-front impacts the boundaries and reflects back to the domain. A larger number of time steps means more interactions inside the domain, resulting in more data recorded in the sensors. However, the memory consumption when using more time steps is higher. Choosing 500 steps was based on a balance between enough recorded data and less computational needs. The initial condition is a small compactly supported Gaussian of the form $Ae^{\frac{-\left((x-x_0)^2+(y-y_0)^2\right)}{2\sigma}}$, with $A = 1$ and $\sigma = 1$. The boundary conditions are of homogeneous Dirichlet type (reflecting).

To train the network we used 54,000 (90%) of the samples as the training set and the remaining 6,000 (10%) as the validation set. We created an additional 15,000 samples for testing the trained model on samples that are completely new. During training, we observed the loss values of the training set and the loss values of the validation set through the training iterations (also called epochs). When both losses declined, the network was training. When the validation loss stopped decreasing, the network is saturating, and when the validation loss started increasing, the network was over-fitting the training data. We saved the model with the lowest validation loss (before saturation or over-fit). We used 400 epochs and a batch size of 32 for training. We also tuned the learning rate of the ADAM [22] optimizer. We chose an initial learning rate of 10^{-3}, and decreased it by half whenever the loss plateaued. We trained the network on a nVidia A6000 GPU for 8 hours. After training, inference takes a fraction of a second (applicable for real-time purposes). An example output of the model, compared to the true obstacles, is shown in Figure 5. From example 1 we observe that the method is able to infer very thin obstacles. From example 2 we see that some of the predictions are very accurate in terms of Intersection Over Union (IOU, defined below). From example 3 we see that the method performs well also for obstacles near the boundary. From example 4 we see that non-convex obstacles have been generated and used in training and testing (this is a test-set example), showing that this method is not constrained by convex obstacles. Example 5 highlights a phenomenon that exists in all other examples as well - we see how the model struggles with pointy shapes, and tries to infer a more smooth edged obstacle. This can be improved with tuning the hyper-parameters of the network, but the current results are satisfactory.

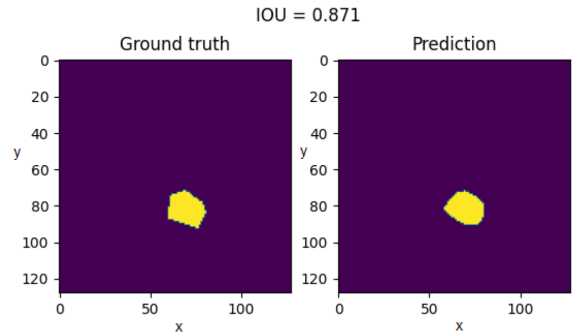
To evaluate the success of the method we used the IOU metric: $IOU = \frac{|\mathcal{O} \cap \tilde{\mathcal{O}}|}{|\mathcal{O} \cup \tilde{\mathcal{O}}|}$, for some sample where the true obstacle image is $\mathcal{O}$ and the prediction is $\tilde{\mathcal{O}}$. We first applied a threshold of 0.5 on $\tilde{\mathcal{O}}$, since it is a probability image. We used this metric to evaluate the predictions on the testing set by checking the mean, median and standard deviation of the IOU over the entire test set. We compared the results to the results in [11], which used a similar approach but did not use the TR method. The results are given in Table 1.

To check the contribution of the PI loss term, we conducted an experiment with 5,000 samples and compared a model trained with the PI loss and one without PI loss. We monitored the validation IOU score during the training and the results clearly indicate that the PI loss network converges faster. However, running the model with the PI loss was computationally heavy and time consuming, so we managed to show the potential on the small data-set. The other experiments were trained without the PI loss.

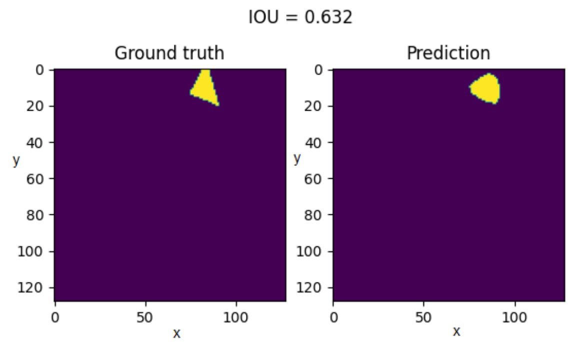
We also analyzed the influence of adding more training data improve the accuracy of the model. As the results in Table 1 clearly indicate, increasing the size of the training data-set produced a more accurate



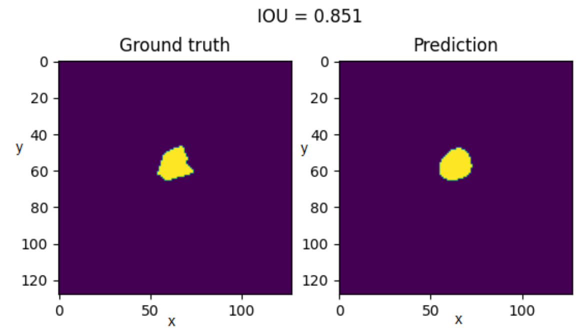
(a) Obstacle identification example 1



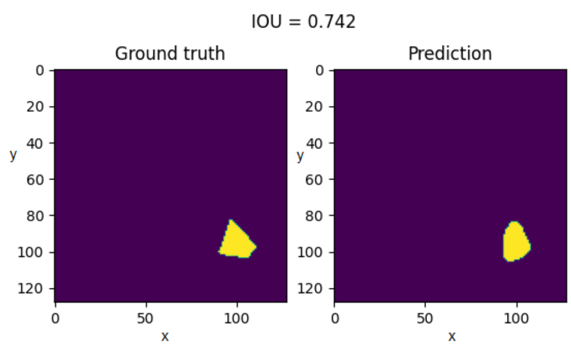
(b) Obstacle identification example 2



(c) Obstacle identification example 3



(d) Obstacle identification example 4



(e) Obstacle identification example 5

Figure 5: Visual examples of outputs of the obstacle identification for different obstacles. Left is the true obstacle image, and right is the network inference output

model. To conduct these experiments we created data-sets consisting of 10,000, 20,000, 30,000 and 60,000 samples, split into 90% training and 10% validation. In addition, we observed that even for the 20,000 samples data-set we already achieved similar results to the benchmark [11].

5 Conclusion

We presented a method for finding the location, shape and size of an underwater obstacle. We used a finite difference scheme to approximate the solution of the wave problem and create a data-set of sensors recordings from many different arbitrary obstacles place inside the domain. We formulated this as a data-driven problem and proposed a method for solving the inverse problem of recovering the obstacle from the sensor measurements. The method we proposed has three blocks: a) using TR to backward propagate the

Method	Number of train samples	Mean IOU	Median IOU	IOU standard deviation
Non-TR [11]	22,500	66%	-	-
Proposed method	9,000	60.96%	67.64%	0.21
Proposed method	18,000	65.88%	72.78%	0.22
Proposed method	27,000	68.21%	75.6%	0.21
Proposed method	**54,000**	**71.69**	**79.27**	**0.21**

Table 1: Statistical aggregation of the obstacle identification results. The metric is the IOU.

solution and recover the initial condition, b) use a deep neural network to extract the obstacle from this reconstruction, and c) a physics-informed loss term that utilizes the forward wave problem to penalize bad predictions made by the network. We show that the method is able to accurately recover the location, shape and size of the obstacle and even compete with the reference methods available in the literature. In this work we did not introduce measurement noise, and we are currently investigating extending this method to work with high measurement noise in the sensors.

References

[1] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[2] Curtis R Vogel. *Computational methods for inverse problems*. SIAM, 2002.

[3] A Tarantola. Inverse problems theory, methods for data fitting and model parameter estimation, 181-188, 1987.

[4] Yurii Ya Belov. Inverse problems for partial differential equations. In *Inverse Problems for Partial Differential Equations*. De Gruyter, 2012.

[5] Rex V Allen. Automatic earthquake recognition and timing from single traces. *Bulletin of the seismological society of America*, 68(5):1521–1532, 1978.

[6] David L Colton, Rainer Kress, and Rainer Kress. *Inverse acoustic and electromagnetic scattering theory*, volume 93. Springer, 1998.

[7] Haiqiang Niu, Emma Reeves, and Peter Gerstoft. Source localization in an ocean waveguide using supervised machine learning. *The Journal of the Acoustical Society of America*, 142(3):1176–1188, 2017.

[8] David L Colton, Richard E Ewing, William Rundell, et al. *Inverse problems in partial differential equations*, volume 42. Siam, 1990.

[9] Albert Tarantola. *Inverse problem theory and methods for model parameter estimation*. SIAM, 2005.

[10] Guofei Pang, Liu Yang, and George Em Karniadakis. Neural-net-induced gaussian process regression for function approximation and pde solution. *Journal of Computational Physics*, 384:270–288, 2019.

[11] Adar Kahana, Eli Turkel, Shai Dekel, and Dan Givoli. Obstacle segmentation based on the wave equation and deep learning. *Journal of Computational Physics*, 413:109458, 2020.

[12] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6), 2021.

[13] Oded Ovadia, Adar Kahana, Eli Turkel, and Shai Dekel. Beyond the courant-friedrichs-lewy condition: Numerical methods for the wave problem using deep learning. *Journal of Computational Physics*, 442:110493, 2021.

[14] Mathias Fink. Time reversal of ultrasonic fields. i. basic principles. *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, 39(5):555–566, 1992.

[15] Mathias Fink and Claire Prada. Acoustic time-reversal mirrors. *Inverse problems*, 17(1):R1, 2001.

[16] Carole H. Sudre, Wenqi Li, Tom Kamiel Magda Vercauteren, Sébastien Ourselin, and M. Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. *Deep learning in medical image analysis and multimodal learning for clinical decision support : Third Inter-*

national Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, held in conjunction with MICCAI 2017 Quebec City, QC,..., 2017:240–248, 2017.

[17] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571, 2016.

[18] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

[19] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111:98–136, 2014.

[20] François Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[21] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.