Adversarial Branch Architecture Search for Unsupervised Domain Adaptation
Supplementary

Luca Robbiano
Politecnico di Torino
Turin, Italy
luca.robbiano@polito.it

Muhammad Rameez Ur Rahman
Sapienza University of Rome
Rome, Italy
rahman@di.uniroma1.it

Fabio Galasso
Sapienza University of Rome
Rome, Italy
galasso@di.uniroma1.it

Barbara Caputo
Politecnico di Torino, CINI Consortium
Turin, Italy
barbara.caputo@polito.it

Fabio Maria Carlucci
Huawei Noah's Ark Lab
London, United Kingdom
fabiom.carlucci@gmail.com

We provide additional analysis, results and implementation details, to further support the claims of the paper and illustrate specific insights. In detail, we present here:

- A more in-depth description of how the Ensemble Model Selection (EMS) module is trained, as well as further implementation details, plots and experimental results regarding it and the different metric it builds on for target accuracy prediction [sec. 1].
- An evaluation of the impact of the search space parameters on the final performance of the UDA method [sec. 2].
- A description of our ResNet50 and ResNet50 + EMS baselines. [sec. 3]
- The full PACS results [sec. 4].

1. Ensemble Model Selection (EMS)

EMS is a crucial component of the ABAS pipeline: in the UDA setting the target labels are not available, and NAS cannot be applied to UDA without a way to assess the quality of the proposed solutions. To be more explicit, EMS uses the metrics discussed in section 3.3 to provide feedback on how good a sample is.

1.1. Training EMS

In this work we build an ensemble of weak predictors by training a linear regressor on top of label-free metrics which weakly correlate with the target accuracy. To do so, we randomly sampled 200 configurations on each dataset and trained them while collecting 100 snapshots for each run, thus ending with a dataset of $20,000$ points. At each snapshot we collected the 6 metrics described in sec. 3.3 thus building a dataset of size 2000×6 on which to train our regressor; note that the pseudo-label metric is only used to assess the best snapshot in a single run, not to compare across runs. All input features were standardized to zero mean and unit standard deviation. To model the regressor we used a linear least-squares model as implemented in Scikit-Learn [5]. As target labels (and thus the corresponding accuracy) is not available in the context of UDA, we trained the regressor on a different dataset and transferred it to the one of interest. Specifically, on Office31 we used the regressor trained on Office-Home and vice-versa. The regressor for PACS was trained on Office31.

1.2. Unsupervised performance estimators

Figures 1, 2 and Table 1 (in aggregated form) show how well the predictors, presented in section 3.3 of the main paper, perform. EMS achieves consistently better correlation on average and future work could investigate if more complex model ensembles could achieve even better performance. Note that since the regressors were only trained on Office31 and Office-Home, Table 2 and the corresponding figures are focused on those two datasets.

1.3. Implementation of the Diversity metric

The original Diversity [8] is defined as:

$$H(\hat{q}(\mathcal{T})) = -\sum_{k=1}^{K} \hat{q}_k log(\hat{q}_k) \tag{1}$$

where K is the number of classes. As we use the Diversity as a predictor for estimating performance across settings with different values of K, we simply divide the metric by K as to get an average over the classes:

$$H(\hat{q}(\mathcal{T})) = -\frac{1}{K}\sum_{k=1}^{K}\hat{q}_k log(\hat{q}_k) \quad (2)$$

Metric	Average corr. with Target Acc. Office31	Office-Home	PACS
Entropy	0.76	0.61	0.57
Diversity	0.76	0.77	0.76
Pseudo-Labels	0.71	0.61	0.68
Source Accuracy	0.42	0.20	0.48
Silhouette	0.70	0.72	0.74
Calinski-Harabasz	0.29	0.03	-0.13
EMS	**0.87**	**0.81**	**0.80**

Table 1: This table shows how the different metrics introduced in section 3.3 of the main paper correlate with the target accuracy. Note that all metrics, with the exception of source accuracy are computed on the target.

2. Impact of the auxiliary branch configuration on the final accuracy

As can be seen in Table 2, the design choices which are part of our search space have a tremendous impact on the final performance of an ALDA [1] training - in other words, the architecture and the hyper-parameters of the auxiliary branch can make or break the UDA approach that builds on it.

3. Baseline implementation

For all our baselines, we use a ResNet-50 backbone pretrained on ImageNet. The DANN and ALDA baselines use the same adversarial branch architecture (two 1024 fully connected layers), connected right after a 512-sized bottleneck layer preceding the ResNet-50 output. The initial learning rate is set to 0.001, and it is adjusted during training following

$$\mu_p = \frac{\mu_0}{(1+\alpha\cdot p)^\beta}, \quad (3)$$

where $\alpha = 10$, $\beta = 0.75$ and p grows from 0 to 1 during training. The weight decay is 0.0005, and the network is trained with SGD (momentum 0.9, batch size 36). Since the bottleneck and adversarial branch are trained from scratch, for those parts of the network we set the initial learning rate to 0.01 and the weight decay to 0.001. For each run (no EMS) we select the last two snapshots (evaluated each 100 iterations) and average them. For the EMS baselines we select the best snapshot of the run according to the value predicted by the EMS regressor. Each run was repeated 4 times and the final accuracies were averaged.

4. PACS results

Table 3 did not fully fit in the main text and is here reported in its entirety. ABAS manages to significantly improve over both DANN and ALDA: surprisingly, on Sketch to Photo, ABAS-DANN largely outperforms ABAS-ALDA, highlighting how an older method can perform better than a new one, when trained with the optimal branch. Indeed, although in a specific setting (P-S), our EMS drives BOHB to select a less than optimal architecture for DANN, ABAS-DANN offers a slightly superior performance overall. Future work to improve EMS would be expected to further increase our results with both methods.

5. Software and hardware

Our network training code is written in PyTorch and based on the publicly available ALDA [1] repository, likewise, we use HpBandSter, the official BOHB [2] implementation. The BOHB algorithm is run for 24 iterations using 8 parallel workers, each running on a Tesla V100 GPU. On average, it takes 6, 10 and 6.4 hours to run the full ABAS pipeline on PACS, Office-Home and Office31 respectively, leading to a cost of 80 GPU hours on the largest setting.

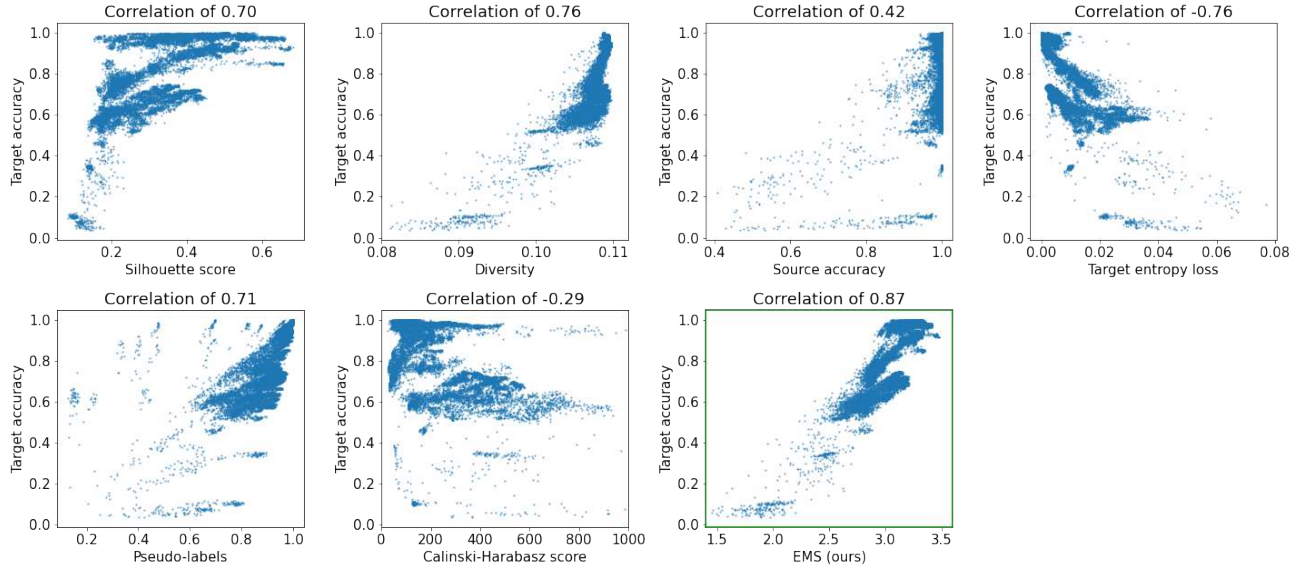

Figure 1: Correlation between the metrics introduced in section 3.3 and the target accuracy, as computed on Office31.

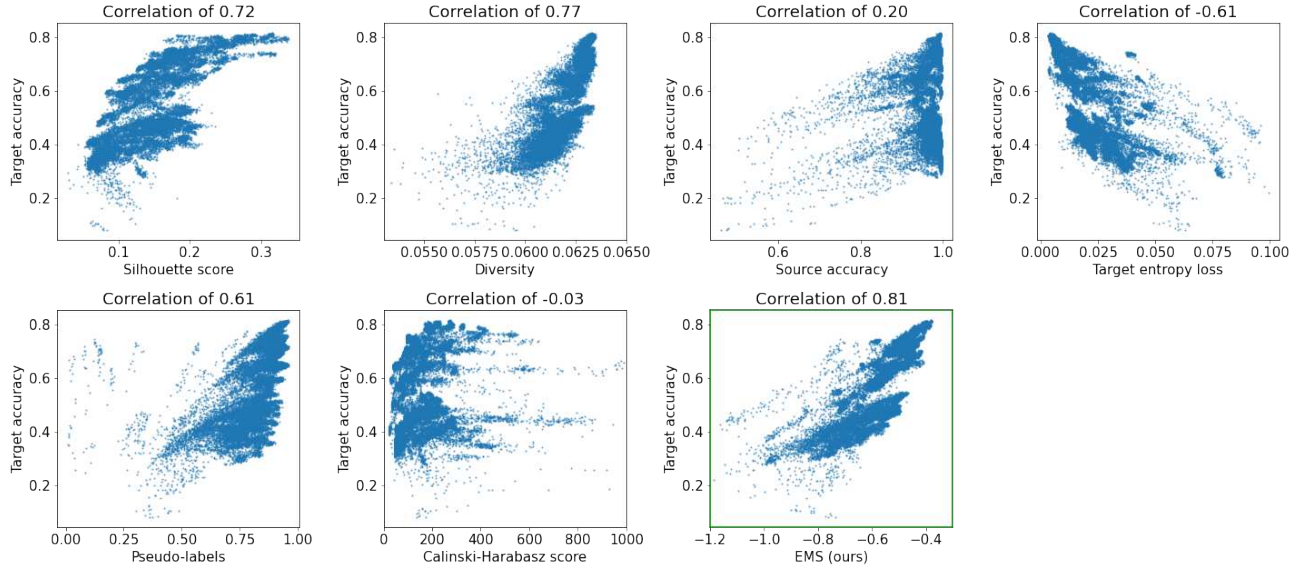

Figure 2: Correlation between the metrics introduced in section 3.3 and the target accuracy, as computed on Office-Home.

Setting	% Div.	Min	Max	Mean ± Std.	Range
A-W	0.23	64.59	95.79	83.4 ± 09.8	31.19
A-D	0.28	72.49	91.57	84.4 ± 06.2	19.08
D-W	0.15	17.11	99.06	89.0 ± 21.7	81.95
D-A	0.28	46.59	70.15	59.3 ± 06.4	23.55
W-D	0.22	16.47	100.00	89.3 ± 24.2	83.53
W-A	0.17	34.97	74.55	63.8 ± 08.3	39.58
Ar-Cl	0.08	26.60	50.71	41.7 ± 06.4	24.11
Ar-Pr	0.14	59.12	65.14	63.2 ± 02.1	06.01
Ar-Rw	0.22	43.30	75.89	70.0 ± 06.3	32.59
Cl-Ar	0.14	36.55	57.87	51.6 ± 07.3	21.32
Cl-Pr	0.14	55.58	66.41	60.9 ± 03.9	10.84
Cl-Rw	0.15	46.12	70.25	63.2 ± 05.6	24.13
Pr-Ar	0.14	40.15	57.21	49.8 ± 04.9	17.06
Pr-Cl	0.11	26.27	49.52	39.7 ± 06.5	23.25
Pr-Rw	0.07	41.47	75.64	66.5 ± 10.7	34.16
Rw-Ar	0.07	59.81	67.86	64.7 ± 02.3	08.06
Rw-Cl	0.07	15.40	54.80	46.2 ± 07.6	39.40
Rw-Pr	0.14	70.30	81.36	77.1 ± 02.6	11.06
AVG	0.2	42.9	72.4	64.6 ± 7.9	29.49

Table 2: Over 400 different points were randomly sampled from our search space and the corresponding models were then trained using ALDA [1] over either Office31 [6] (top 6 rows) or Office-Home [7] settings. Note how the configuration of the auxiliary branch significantly affects the model performance. *% Div.* stands for the fraction of runs which did not converge - *Min* is the minimum accuracy - *Max* is the maximum - *Mean* reports the mean ± the standard deviation, and *Range* is the total range of accuracies, defined as $Max - Min$.

	P-A	C-A	S-A	A-P	C-P	S-P	A-C	S-C	P-C	A-S	C-S	P-S	AVG
ResNet-50 [4]	62.3	71.2	30.8	96.6	89.7	38.0	57.0	53.5	27.6	43.0	59.1	30.7	55.0
ResNet-50 + EMS	62.0	72.0	25.6	97.7	90.9	38.8	57.5	53.2	23.0	41.3	58.7	28.2	54.1
DANN [3]	79.9	88.4	70.3	97.3	95.1	57.1	82.4	70.8	64.7	62.5	68.7	61.4	74.9
DANN + EMS	80.3	89.0	72.6	98.1	95.6	56.2	82.2	73.3	64.0	63.4	69.7	59.6	75.3
ABAS-DANN	91.7	90.7	68.2	98.3	96.2	**82.8**	87.2	71.2	73.7	68.7	76.0	47.2	**82.9**
ALDA [1]	89.3	91.9	69.9	98.3	97.3	63.4	85.1	75.2	**74.3**	79.2	70.6	60.7	79.6
ALDA + EMS	90.2	**92.0**	72.3	98.4	**97.8**	69.5	86.0	82.1	72.1	**80.7**	**75.1**	**66.1**	81.9
ABAS-ALDA	**93.1**	91.8	**78.1**	**98.7**	**97.8**	70.8	**88.7**	**84.9**	69.7	79.8	69.5	64.9	82.3

Table 3: Results of ABAS on PACS, using a ResNet-50 backbone, + EMS: experiments run with our model selection strategy.

References

[1] Minghao Chen, Shuai Zhao, Haifeng Liu, and Deng Cai. Adversarial-learned loss for domain adaptation. In *AAAI*, pages 3521–3528, 2020.

[2] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

[3] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010.

[7] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5018–5027, 2017.

[8] Xiaofu Wu, Quan Zhou, Zhen Yang, Chunming Zhao, Longin Jan Latecki, et al. Entropy minimization vs. diversity maximization for domain adaptation. *arXiv preprint arXiv:2002.01690*, 2020.