Adversarial Branch Architecture Search for Unsupervised Domain Adaptation

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

Abstract

Unsupervised Domain Adaptation (UDA) is a key issue in visual recognition, as it allows to bridge different visual domains enabling robust performances in the real world. To date, all proposed approaches rely on human expertise to manually adapt a given UDA method (e.g. DANN) to a specific backbone architecture (e.g. ResNet). This dependency on handcrafted designs limits the applicability of a given approach in time, as old methods need to be constantly adapted to novel backbones.

Existing Neural Architecture Search (NAS) approaches cannot be directly applied to mitigate this issue, as they rely on labels that are not available in the UDA setting. Furthermore, most NAS methods search for full architectures, which precludes the use of pre-trained models, essential in a vast range of UDA settings for reaching SOTA results.

*To the best of our knowledge, no prior work has addressed these aspects in the context of NAS for UDA. Here we tackle both aspects with an Adversarial Branch Architecture Search for UDA (ABAS): **i.** we address the lack of target labels by a novel data-driven ensemble approach for model selection; and **ii.** we search for an auxiliary adversarial branch, attached to a pre-trained backbone, which drives the domain alignment. We extensively validate ABAS to improve two modern UDA techniques, DANN and ALDA, on three standard visual recognition datasets (Office31, Office-Home and PACS). In all cases, ABAS robustly finds the adversarial branch architectures and parameters which yield best performances. https://github.com/lr94/abas.*

1. Introduction

Unsupervised Domain Adaptation (UDA) enables the transfer of domain knowledge from a source domain to a target domain, for which possibly few data are available, but no target labels. Performance has increased steadily in recent years [4, 15, 48, 25, 47], but domain transfer remains constrained to specific setups [10, 13, 24]. In other words, whenever a new architectural backbone becomes available, optimally adapting existing UDA methods to it requires a manual and slow process of trial-and-error.

While Neural Architecture Search (NAS) has recently made large progress [41, 20, 40, 33] in removing the heuristic component of neural architecture design, it does not directly apply to UDA for two main reasons: **i.** NAS requires target labels for model validation when searching for the optimal architecture, which are not available in UDA; **ii.** the vast majority of UDA methods applied to visual recognition tasks, from classification to detection up to segmentation, require the use of pre-trained (backbone) models to reach state-of-the-art performance on real world datasets [5, 4, 48, 47]. To the best of our knowledge, no research has addressed these essential aspects in the context of NAS for UDA.

We propose a novel Adversarial Branch Architecture Search (ABAS - Fig. 1) for UDA to address both limitations. Our work builds on most recent and robust UDA methods which use *auxiliary adversarial branches* [8, 2, 4, 39]: a secondary network (the adversarial branch) is attached to a main backbone (see Fig. 2 for an illustration). While the main network is trained supervisedly on the source images and labels, the secondary branch reduces the distributional gap between the source and target domains, without target labels, by adversarially making the domains indistinguishable. The architecture of the auxiliary branch, which plays a huge role in the final accuracy of the method (as shown in Fig. 5), has so far been hand-crafted.

In ABAS, as first main contribution, we address the lack of target labels by a novel data-driven Ensemble approach
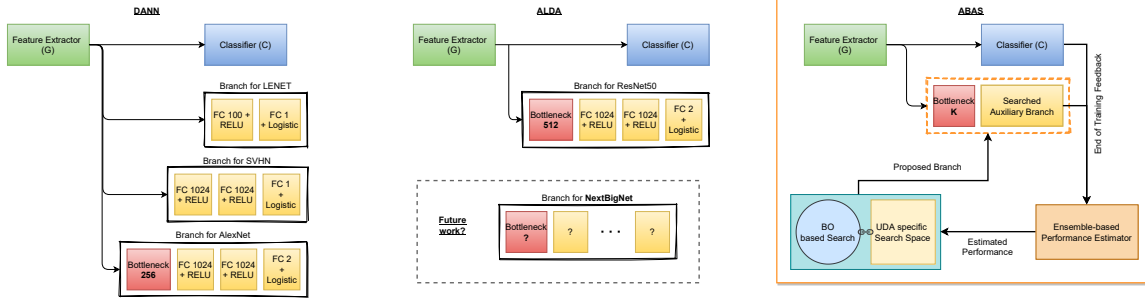
Figure 1: The original DANN [8] paper proposed a specific implementation for each of the considered backbone architectures. Later authors (e.g. ALDA [4]) empirically adapted the core idea to different architectures through a process of trial and error. Whenever a new architecture is proposed, this tedious procedure needs to be repeated (note this issue is shared by any other UDA method based on an auxiliary branch). To address this, in ABAS we propose to automatically search for the optimal architecture of the auxiliary branch in a data driven way. Through ABAS, it is possible to seamlessly apply DANN to any future architecture without the need for human supervision. What follows is a brief description, more details can be found in Algorithm 1. At each iteration, the Bayesian Optimization (BO) acquisition function proposes a number of candidate branch architectures. After training, we extract a number of supervised (source) and unsupervised (target) features which are fed into our ensemble-based performance estimator. Its feedback is sent to BO, thus closing the loop.

for Model Selection (EMS), which directly applies to target data. Model selection in UDA is challenging and often overlooked. Only very few works have so far focused on it [51, 26], but they only consider source features. By contrast, considering the actual target features yields better domain transfers, as we show.

As a second main contribution we propose using a multi-fidelity Bayesian Optimization (BOHB) [7] method to search for both the architecture and the hyper-parameters of the auxiliary branch. ABAS exploits the backbone architecture and pre-training, which is key to surpass the current state-of-the-art performance.

We conduct an in-depth performance evaluation on a popular UDA technique, DANN [8], and on a more recent state-of-the-art method, ALDA [4]. We test their ABAS extensions on three computer vision datasets, PACS [14], Office31 [35] and Office-Home [44]. Our study confirms that the default DANN and ALDA branches perform well on specific backbones and source-target cases, i.e. for Office31 and Office-Home, well above random architectures (cf. Fig. 5 and Sec. 4). However, they underperform on PACS, where ABAS also finds the best UDA architecture.

2. Related work

Unsupervised Domain Adaptation (UDA). Research on UDA has been recently divided into four families of methods [54]: discrepancy, adversarial discriminative, adversarial generative and self-supervision based methods. Adversarial discriminative approaches provide top performance, but they are significantly influenced by the specific architecture of the auxiliary branch, cf. Fig. 5: un-optimized architectures may diverge and/or yield a wide span of accuracies. We find this strongly motivates research on architec-
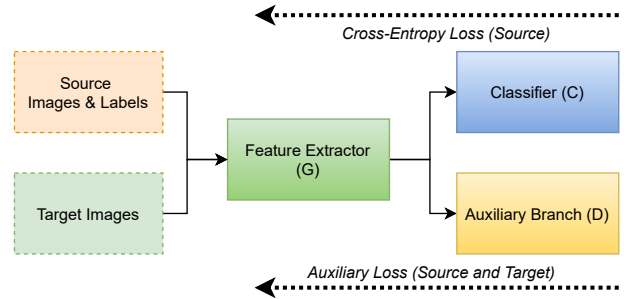

Figure 2: Many UDA methods fall under this general framework. The auxiliary branch D can compute either a domain discriminator [8, 4], or a self-supervised [2, 39] loss.

ture search for these approaches.

Starting from [8], UDA has been cast as the problem of learning domain-agnostic features via an auxiliary branch and an adversarial training. The extra branch is attached to the backbone and tasked to learn to discriminate the domains. Then its gradient is reversed to make the backbone features indistinguishable between the source and the target domain. A few recent improvements are notable: [12] considers the cross-covariance between samples; [21] takes the classifier prediction into account; [42] generalizes a similar approach to the semi-supervised setting; [3] extends the discriminative gradient reversal to the pixel level; and finally [4] combines the adversarial training with self-training. All of the above propose specific embodiments of the auxiliary adversarial branch, but offer no guidelines on how to adapt it to different backbones. To the best of our knowledge, this is the first work proposing a data-driven pipeline to learn it.

Model Selection for UDA. Hyper-parameter and model selection are extremely challenging in the UDA setting, as

Hyperparameters	Ranges	Type
Domain adaptation weight λ	$[0, 2]$	real
Dropout Probability	$[0, 1]$	real
# FC layers	$[1, 6]$	integer
Bottleneck size fc_b	$[2^6, 2^{10}]$	integer
Hidden units size fc_h	$[2^6, 2^{12}]$	integer

Figure 3: Left: overview of how the ABAS search space applies to the main architecture (BS is batch size). Right: explicit search space definition. Being low-dimensional it can easily be optimized through BO; on the other hand (as shown in Fig. 6) each of these parameters has a strong effect on the final accuracy.

target labels are not available. In practice, many works simply fix the hyper-parameters across a range of datasets [43, 29, 36, 34].

A more principled approach based on importance-weighted cross-validation (IWCV) was first proposed by [38]: the more the source sample is similar to the target, the higher the importance. The IWCV estimator is unbiased, but has unbounded variance; Deep Embedded Validation [51], was proposed to reduce the variance and adapt IWCV to deep features. Similarly, [26] proposes a reduced-variance IWCV estimator for hyper-parameter optimization in the context of multi-source DA. The current literature focuses on the source, only implicitly taking target features into consideration. By contrast, we propose a performance estimator which mostly uses the actual target features (see Sec. 3.3).

Neural Architecture Search (NAS) and UDA-NAS. There is a wealth of literature on NAS [56] leveraging diverse search strategies: Bayesian Optimization (BO), evolutionary, reinforcement learning, and gradient-based, as recently surveyed in [6, 30]. BO is one of the most efficient strategies for sample-based hyper-parameter tuning [37], and has been recently applied to NAS [45, 50]. Multi-fidelity BO (BOHB) [7], which further reduces the search cost by performing partial evaluations, has also been successfully applied to NAS [52, 33]. ABAS exploits pre-trained backbones and efficiently fine-tunes the architectures, enabling us to use a more accurate sample-based technique. In particular, we adopt BOHB for its sota performance.

A recent pre-print [17] has proposed a DARTS-like [20] method for UDA. While that shares similar motivations with ABAS, it is unable to leverage pre-trained models and as such it can only be applied to simple low-resolution datasets. Furthermore model selection is performed by a simple validation on the source domain, which we show to be insufficient in the general case.

3. Method

We propose a data-driven pipeline to search for the optimal architecture of the auxiliary branch in the context of unsupervised domain adaptation. As mentioned in section 2, a number of UDA approaches require the use of an auxiliary branch: while we explicitly explore how this idea applies to works based on domain adversarial learning, it is trivially extendable to other methods (see section 5). Here we apply ABAS to 1) the first UDA adversarial approach, DANN [8] and to 2) the current state-of-the-art variation, ALDA [4]. In both cases, the auxiliary branch is adversarially trained to reduce the distributional gap at the feature level.

3.1. Background

Due to space constraints, we provide here only a mostly intuitive overview and refer to the referenced papers for a formal presentation of background methods.

Adversarial branches for domain adaptation Let us define a dataset $\boldsymbol{X}_s = \{\boldsymbol{x}_s^i, y_s^i\}_{i=0}^{N_s}$ drawn from a labeled source domain $\mathcal{S}$, and a dataset $\boldsymbol{X}_t = \{\boldsymbol{x}_t^j\}_{j=0}^{N_t}$ from a different unlabeled target domain $\mathcal{T}$, sharing the same set of categories. Our goal is to maximize the classification accuracy on $\boldsymbol{X}_t$ while training on $\boldsymbol{X}_s$. As it can be seen in Figure 2, the overall architecture consists of 3 components: the feature extractor G, the labeled classifier C and the adversarial branch D. The basic intuition behind discriminative adversarial approaches for domain adaptation is that a domain classifier is trained to distinguish between source and target samples; as its gradient is reversed (Fig. 3), the features in G are trained so that source and target have similar representations. The model is trained by optimizing the following objective function:

$$\min_{G,C,D} \mathcal{L}_{CE}(C(G(x_s)), y_s) + \lambda\mathcal{L}_{Adv}(D(G(x_s \cup x_t))) \quad (1)$$

Where $\mathcal{L}_{CE}$ is the standard cross-entropy loss, computed on source samples only, and $\mathcal{L}_{Adv}$ is the adversarial loss trained to reduce the domain gap. In the case of DANN [8],

$$\mathcal{L}_{Adv}(D(G(x_i)), d_i) = d_i \frac{1}{D(G(x_i))} + (1-d_i) \frac{1}{1 - D(G(x_i))}$$ (2)

With d_i being a binary variable (domain label) indicating whether the sample i belongs to the source or to the target.

ALDA [4], is a modern DANN variant, currently state-of-the-art, which combines domain adversarial training with self-training. While its overall loss contains multiple terms, we focus here only on the adversarial loss:

$$\mathcal{L}_{Adv}(x_s, y_s, x_t) = \\ \mathcal{L}_{BCE}(c^{(x_s)}, y_s) + \mathcal{L}_{BCE}(c^{(x_t)}, u^{(\hat{y}_t)})$$ (3)

L_{BCE} is the Binary Cross-Entropy loss, $c^{(x)}$ is the corrected pseudo-label vector generated by multiplying the confusion matrix and the pseudo-label vector $\hat{y}$, and $u^{(\hat{y}_t)}$ is the opposite distribution of $\hat{y}_t$. Intuitively, the authors propose to use a discriminator branch to predict the confusion matrix instead of the domain probabilities. The confusion matrix is then used to correct the pseudo-label vector obtained by feeding target samples into the network, which is then used as the target label for the training process. The discriminator is trained to produce different corrected vectors for each domain, while the feature extractor learns to fool it. ABAS will be used to search for the best performing architecture for D in both DANN and ALDA.

Bayesian Optimization for AutoML Several optimization strategies could be used to guide the search, but our search space is compact and has low dimensionality (cf. Sec. 3.2), which makes it particularly suitable for Bayesian Optimization (BO). For efficiency, we adopt BOHB, a multi-fidelity combination of BO and Hyperband [16]. While single fidelity BO evaluates all samples with full budget, BOHB resorts to partial evaluations with smaller-than-full budget, excluding bad configurations early in the search process and reserving computational resources for the most promising configurations. So, given the same time budget, it evaluates many more configurations and it achieves faster optimization than competing methods. Details of the BOHB's role in ABAS are reported in Sec. 3.4

3.2. ABAS Search space

UDA methods for visual object categorization require the use of a pre-trained model (typically on ImageNet) to be competitive. Hence, the architecture of G and C are usually fixed as a subset of the backbone: in practice the only choice being made is where G ends and C begins (see Fig. 2). Our search space (Fig. 3) is a combination of hyper and architectural parameters: the domain loss weight λ (eq. 1) and the dropout probability of the fully connected layers in D are hyper-parameters, while the number of fully

connected layers (#fc layers), size of the hidden units (*fc_h*) and of the bottleneck (*fc_b*) fully characterize the width and the height of the auxiliary branch D. Giving the bottleneck layer a separate width allows to control the overall capacity of the auxiliary branch by limiting how much information it is fed.

3.3. Ensemble-based Model Selection (EMS)

Model selection in the context of UDA is extremely challenging, as target labels are not available. To overcome this, we propose using an ensemble of weakly correlating predictors. Experimentally, we show how a linear regressor over these metrics, trained on one dataset, is capable of capturing the ranking between models on a *different* unseen dataset, thus highlighting the generalization power of this approach.

A description of the considered metrics follows ([S] or [T] implies on which domain the metric is computed). 1. **Entropy** [T] [9] minimization is commonly used in UDA methods as a regularizer for unlabeled samples [32]; intuitively a low entropy corresponds to well separated classes. Mathematically the target entropy is defined as $H(C(G(X_T))) = -\frac{1}{N_t} \sum_{x^i \in X_T} C(G(x^i)) log(C(G(x^i)))$. 2. **Diversity** [T], recently proposed by [46], measures the diversity of class predictions in a batch. It is defined as the entropy of the predicted category distributions, $H(\hat{q}(\mathcal{T}))$. 3. **Silhouette and Calinski-Harabasz score** [T] [31, 1]: the most confident label is assigned to the target samples, which are then clustered with K-Means using the predicted class centroids. The Silhouette and Calinski score can then be used to estimate how well separated are the different classes. 4. **Source (weighted) loss** [S]: the cross-entropy loss can be computed on the labeled source samples to estimate the performance of the predictor. Additionally it is possible to weight the samples according to their similarity to the target, as in Importance Weighted Validation [38, 51]. We initially experimented with the source-weighted accuracy in EMS, but due to a number of practical drawbacks we decided to adopt the vanilla source accuracy instead. Specifically, the performance of Importance Weighting methods heavily depends on how well the regression model estimates the density ratio, which is not trivial to successfully implement; secondly, it requires a separate source validation set (not always available on small settings such as Office31). 5. **Time consistent pseudo-labels** [T]: inspired by [55], we propose to use the mode of all model predictions during training (one prediction per epoch) as target pseudo-labels, as the label predicted more often is likely to be the true one. Once training has ended, we compute these pseudo-labels and use them to retroactively estimate the target accuracy at each epoch.

Note how all of these metrics, with the exception of the source accuracy, are computed on the target domain. These metrics are all relatively cheap to compute and have good
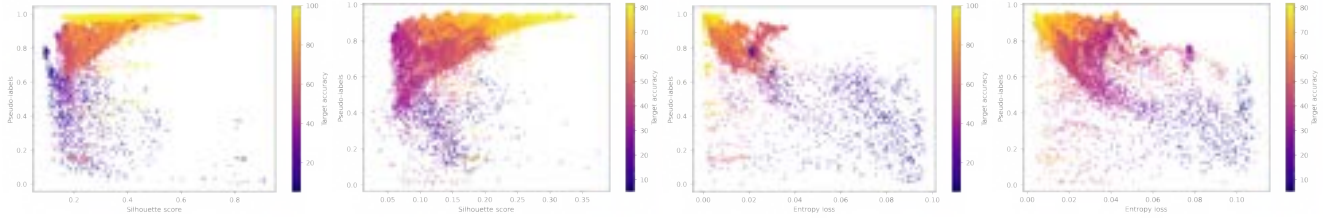
Figure 4: Different model configurations trained on Office31 (1st and 3rd) and Office-Home (2nd and 4th). The x and y axes show two different unsupervised validation metrics we use in EMS, while the color represents target accuracy. Clearly, by only using a single metric it is extremely hard to uniquely predict the performance, but the combined use of two metrics makes it easier to distinguish between good and bad models. This shows how different metrics can be complementary and employed to build a better estimator of model performance. The similarity between pairs of plots on different settings (1&3 and 2&4) suggests that an estimator built on one dataset can be successfully transferred on another. Further plots can be found in the Supplementary.

correlation on their own with the target accuracy, but as we show in Fig. 4 and in the Supplementary, they are all individually vulnerable to degenerate cases. To overcome this, we learn a simple least-squares linear regressor to predict the target accuracy. Since it is not possible to use the target labels of interest we instead learn the regressor on one experimental setting and then use it on a different one. It is worth mentioning that although the output of the regressor cannot be used to estimate the *accuracy* of a model on an unseen dataset, its output has a strong correlation with the ground-truth ranking, as we show in section 4.1. We use this predictor to not only return the best sample, but also the best snapshot during training.

3.4. Auxiliary branch optimization

A full overview can be found in Fig. 1 and Algorithm 1: ABAS combines BOHB [7] and an adversarial method of choice (sec. 3.1), with our UDA specific search space and model selection strategy. Given a fixed budget, ABAS performs a number of rounds, alternating between sampling and evaluating. At each step, we sample the Bayesian acquisition function $\alpha(\Theta|D)$ for B different configurations $\{\Theta_t^j\}_{j=1}^B$. The configurations are used to build auxiliary branches for method Q and the resulting network is trained on the target setting. After training, supervised (from the source) and unsupervised (from the target) features are collated for our model selection module, as described in section 3.3. The ensemble predictor finally gives feedback to the BO process. This procedure is repeated for a given number of rounds and in the end the best performing model is returned. The pipeline itself is relatively simple and, as shown in the next section, capable of significantly improving over existing sota methods. Finally, while we decided to use BOHB in this particular implementation, it would be feasible to use a different search strategy.

Algorithm 1 BO auxiliary branch optimization

1: **Input:** Domain adaptation method Q, Performance estimator E, BO surrogate model $p(f|\Theta, D)$ and acquisition function $\alpha(\Theta|D)$
2: **for** $t = 1$ **to** T **do**
3:     Recommend $\{\Theta_t^j\}_{j=1}^B = \arg\max \alpha_{t-1}(\Theta|D)$
4:     **for** $j = 1$ **to** B **in parallel do**
5:         Build the auxiliary branch and evaluate $E(Q(\Theta_t^j))$ to obtain its corresponding performance metric f_t^j
6:     **end for**
7:     Update D and thus $p(f|\Theta, D)$ with $\{\Theta_t^j, f_t^j\}_{j=1}^B$
8: **end for**
9: **return** The best model according to $E(Q(\Theta^*))$
---

4. Experiments

Datasets. We evaluate our method on three publicly-available multi-domain datasets. The **Office31** dataset [35], composed of $4,652$ images, is a standard benchmark for domain adaptation. The images depict objects belonging to 31 categories and taken from three different domains: Amazon (A), DSLR (D), and Webcam (W). Webcam and DSLR pictures were manually captured in the same office environment with a consumer webcam and a reflex camera. Amazon samples are images of products downloaded from amazon.com. The **Office-Home** dataset [44] contains objects from 65 categories and a total of around $15,500$ images from four domains: Art, Clipart, Product, and Real-World. Like Office31's Amazon domain, the Product domain features product pictures taken from websites. Real-world images were captured with a regular digital camera. **PACS** [14] is composed of $9,991$ images from four domains, each containing objects from 7 categories. While PACS is commonly used for domain generalization, we employ it in our single-source setting to explore a diverse test-bed.
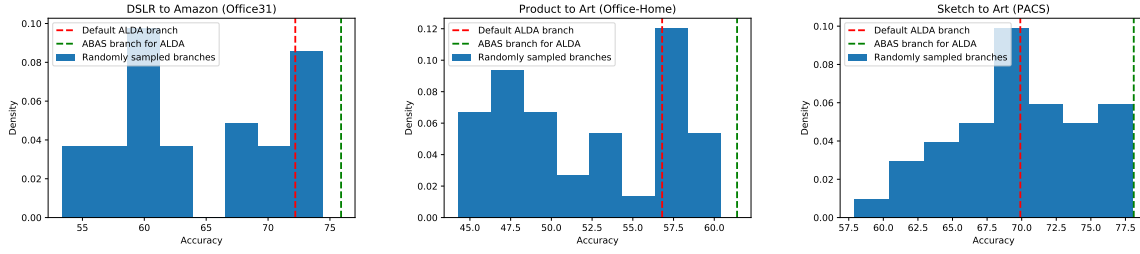
Figure 5: Accuracy histograms obtained by randomly sampling 40 architectures for ALDA's [4] auxiliary branch and training them on three settings (10% didn't converge and were excluded). Contrary to many NAS search spaces [49], ours is large enough to offer a significant impact on final performance (around 20%) on all settings. Equally important, while the default ALDA branch performs well on Office31 and Office-Home (the datasets it was tested on), it performs significantly worse on PACS. ABAS, on the other hand, can adapt itself to all settings, consistently outperforming the random sampling baseline. See sec. 4.1 for full details.

Baseline methods To highlight the improvements brought by our ABAS pipeline, we apply it to two representative UDA methods: DANN [8], the original domain adversarial approach, to this day and base on which all methods of this family build on, and ALDA [4] the current state-of-art among discriminative adversarial approaches. **Training protocol** ABAS is based on an outer BOHB loop alternated with an inner loop in which different networks are trained. For BOHB, we set $\eta = 3$, minimum budget 2000 iterations, maximum budget 6000 iterations and run it for 24 iterations. Intuitively, the most promising solutions evaluated at a lower budget will be re-run at a higher budget. For each sample, BOHB indicates a computational budget and an architectural configuration, which are used to build the appropriate auxiliary branch in the context of the adversarial method of choice.

In all experiments we use a ResNet50 [11] backbone pretrained on ImageNet; the adversarial branch (built according to each specific sample, see sec. 3.2) is located immediately after a bottleneck layer preceding the final classification layer (Fig. 3). Each model is trained end to end, using the same training code and protocol as [4]: on all settings, we use SGD optimizer, batch size 36, momentum 0.9, weight decay 0.0005 and initial learning rate of 0.001. We adjust the learning rate by $\mu_p = \frac{\mu_0}{(1+\alpha \cdot p)^\beta}$, where p is the training progress linearly changing from 0 to 1, $\mu_0 = 0.001$, $\alpha = 10$ and $\beta = 0.75$, following [8], the weight of the gradient reversal layer is gradually increased from 0 to 1: $\rho = \frac{2}{(1+exp(-\gamma \cdot p))} - 1$ where γ is set to 10 for all experiments. Since the bottleneck and the auxiliary branch are trained from scratch, we use a higher learning rate ($10\mu_p$) and weight decay (0.001) for those parts of the network. At each iteration, a source and target batch are fed into the backbone. The cross-entropy loss is computed on the source, while both source and target samples contribute to the adversarial branch loss (DANN or ALDA loss, de-

pending on the setting). At the end of training, both source and target features are then fed into the performance estimator (as detailed in section 3.3) to provide feedback to the Bayesian Optimization process.

To train our regressors, we randomly sampled 200 configurations on each dataset and trained them while collecting 100 snapshots for each run (total of $20,000$ data points). At each snapshot we collected the metrics described in sec. 3.2 thus building a dataset to train our regressor on; note that we only use the pseudo-label metric 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. For the regressor we used a linear least-squares model as implemented in Scikit-Learn [27]. On Office31 we used a regressor trained on Office-Home and vice-versa. The PACS regressor was trained on Office31.

4.1. Preliminary analysis

Search space expressivity and sensitivity. Given the added cost of automatically searching for the optimal architecture, it is important to understand whether 1) the design of the adversarial branch has an impact on the final accuracy and 2) if a "good" architecture is actually data-dependent. To assess this, we followed NAS best practices [49] and randomly sampled 40 configurations on each setting to evaluate our search space. We observed the following: the design of the adversarial branch can make or break a training. Even excluding a significant (10%) portion of models which failed to converge, on average the design of the adversarial branch accounted for at least 20% in absolute performance (Fig. 5 - full results in the supplementary) - note that λ was fixed to focus on the architecture only. Furthermore, as shown in Fig. 6, the "best" configuration heavily depends on the specific setting being considered. Indeed, while the adversarial branch proposed in ALDA performs well on Office31 and Office-Home (the datasets it was originally tested on), it significantly lags behind on several PACS settings
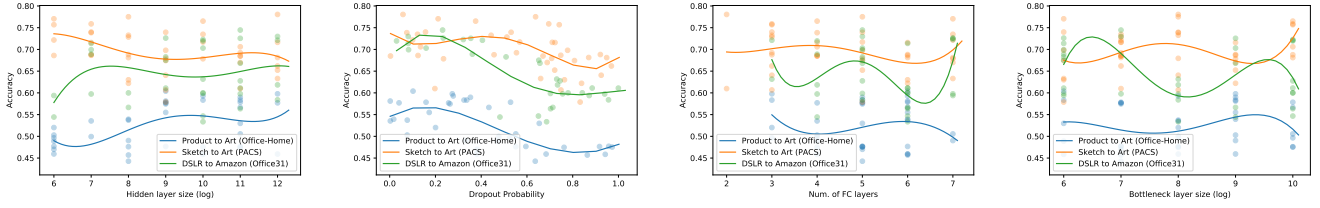
Figure 6: Search space sensitivity analysis. We randomly sampled 40 configurations and trained them on a representative setting for each of the considered datasets (Office31, Office-Home, PACS). It is evident that the optimal value changes significantly across tasks. X-axis represents the search space hyper-parameter with respect to which the plot is computed.

(Fig. 5 and Table 3).

Performance estimation. As discussed in sec. 2, and recently highlighted by [10], strong model selection should be a key element of any transfer learning approach; more so when proposing a UDA method capable of explicitly tuning the model hyper-parameters themselves. In our next experiments we analyzed how well different metrics correlate with the target accuracy. We sampled 40 configuration on each setting and computed the metrics discussed in section 3.3 at each epoch of training. Full overview on the results can be found in the supplementary, while representative plots are shown in Fig. 4. Note how, individually, all metrics perform as weak predictors, correlating with the target accuracy but failing in specific instances. By considering a linear combination we provide a much stronger estimate.

To assess our model selection strategy in isolation from the NAS component, we retrained vanilla ALDA [4] and reported the performance according to either the last epoch (as done in the paper) or according to our model selection (Tables 1,2,3). It is worth mentioning that a) we could not replicate their results (even though we used the official code) and b) on Office-Home and PACS our model selection improved average results by up to 2 full p.p.

4.2. Comparison with SOTA

Comparison with current sota methods can be found in tables 1 to 3. ABAS significantly improves over the UDA methods it builds on (even when they use our same model selection strategy), with both DANN and ALDA and on all three datasets. The smallest average improvement we observe is 1.9% (Office31 - ALDA 87.5 vs ABAS-ALDA 89.4) while the largest is 3.1% (Office-Home - DANN 56.0 vs ABAS-DANN 59.1). With the exception of Office-Home, where we couldn't replicate the official ALDA results, ABAS-ALDA is the new sota on all settings. It is worth mentioning that on one of the easiest Office31 settings (D-W), ABAS-ALDA is outperformed by a few different methods; we speculate that, since DSLR and Webcam are very close, the adversarial branch fails to capture meaningful differences. Indeed, ALDA similarly performs poorly on this setting.

5. Discussion

The shift from hand-crafted to principled architecture search is ongoing, as it is witnessed by the flourishing NAS research and techniques. ABAS fills in two important requirements of NAS for UDA: it provides a data-driven strategy (EMS) for model selection, circumventing the lack of target labels; and it focuses on searching the architecture of auxiliary branches attached to a pre-trained backbone, essential practise for state-of-the-art performance.

Only most recent research has stressed the importance of model selection for UDA [10]. The lack of a validation identically distributed as the test set, led researchers to indulge on selecting hyper-parameters on the test set. Hopefully the propositions here (cf. EMS) would be initial steps towards more methodologically sound model selection practises.

Finally, we emphasize that ABAS provides tools (search space, EMS, branch optimization), which comply with the use of most other NAS methods and potentially all UDA techniques based on an auxiliary branch. A notable example is self-supervision-based UDA methods using an auxiliary branch. Since they adopt the same exact macro-architecture as in Fig. 2, then ABAS seamlessly applies there.

Acknowledgments

	A-W	A-D	D-W	D-A	W-D	W-A	AVG
ResNet-50 [11]	68.4±0.2	68.9±0.2	96.7±0.1	62.5±0.3	99.3±0.1	60.7±0.3	76.1
DANN [8]	82.0±0.4	79.7±0.4	96.9±0.2	68.2±0.4	99.1±0.1	67.4±0.5	82.2
ADDA [43]	86.2±0.5	77.8±0.3	96.2±0.3	69.5±0.4	98.4±0.3	68.9±0.5	82.9
JAN [23]	85.4±0.3	84.7±0.3	97.4±0.2	68.6±0.3	99.8±0.2	70.0±0.4	84.3
MADA [28]	90.0±0.1	87.8±0.2	97.4±0.1	70.3±0.3	99.6±0.1	66.4±0.3	85.2
CBST [57]	87.8±0.8	86.5±1.0	98.5±0.1	71.2±0.4	**100±0.0**	70.9±0.7	85.8
CAN [53]	92.5	90.1	98.8	72.1	**100.0**	69.9	87.2
CDAN+E [22]	94.1±0.1	92.9±0.2	98.6±0.1	71.0±0.3	**100.0±0.0**	69.3±0.3	87.7
MCS [18]	75.1	71.9	96.7	58.8	99.4	57.2	76.5
DMRL [47]	90.8±0.3	93.4±0.5	99.0±0.2	73.0±0.3	**100.0±0.0**	71.2±0.3	87.9
DM-ADA [48]	83.9±0.4	77.5±0.2	**99.8±0.1**	64.6±0.4	99.9±0.1	64.0±0.5	81.6
3CATN [15]	95.3±0.2	94.1±0.3	99.3±0.5	73.1±0.2	**100±0.0**	71.5±0.6	88.9
ALDA [4]	95.6±0.5	94.0±0.4	97.7±0.1	72.2±0.4	**100.0±0.0**	**72.5±0.2**	88.7
ResNet-50	67.0±1.9	76.4±1.4	94.8±1.3	56.2±1.0	98.7±0.1	58.3±0.9	75.2
ResNet-50 + EMS	70.4±2.0	77.1±2.0	95.6±0.6	58.2±2.6	99.1±0.5	60.1±1.1	76.8
DANN	85.0±0.5	82.5±0.5	96.7±0.2	63.9±1.1	99.2±0.3	64.7±0.7	82.0
DANN + EMS	85.5±2.1	82.8±2.2	97.1±1.0	64.1±0.5	99.8±0.2	64.5±2.0	82.3
ABAS-DANN	89.4	87.6	98.4	64.1	99.8	69.3	84.8
ALDA	94.8±0.6	91.6±0.9	98.3±0.4	69.6±0.9	99.9±0.0	70.8±0.7	87.5
ALDA + EMS	95.0±1.4	91.9±0.7	98.6±0.4	69.3±2.1	99.7±0.3	71.1±0.8	87.6
ABAS-ALDA	**96.1**	**95.0**	98.5	**75.9**	**100.0**	70.7	**89.4**

Table 1: Results of ABAS on Office31 with a ResNet-50 backbone. + EMS: experiments run with our model selection.

	Ar-Cl	Ar-Pr	Ar-Rw	Cl-Ar	Cl-Pr	Cl-Rw	Pr-Ar	Pr-Cl	Pr-Rw	Rw-Ar	Rw-Cl	Rw-Pr	AVG
ResNet-50 [11]	34.9	50.0	58.0	37.4	41.9	46.2	38.5	31.2	60.4	53.9	41.2	59.9	46.1
DANN [8]	45.6	59.3	70.1	47.0	58.5	60.9	46.1	43.7	68.5	63.2	51.8	76.8	57.6
JAN [23]	45.9	61.2	68.9	50.4	59.7	61.0	45.8	43.4	70.3	63.9	52.4	76.8	58.3
CDAN+E [22]	50.7	70.6	76.0	57.6	70.0	70.0	57.4	50.9	77.3	**70.9**	**56.7**	81.6	65.8
TAT [19]	51.6	69.5	75.4	59.4	69.5	68.6	59.5	50.5	76.8	**70.9**	56.6	81.6	65.8
ALDA [4]	**53.7**	70.1	**76.4**	**60.2**	**72.6**	**71.5**	56.8	**51.9**	77.1	70.2	56.3	**82.1**	**66.6**
ResNet-50	35.6	61.5	70.4	44.0	56.1	57.4	46.5	31.9	69.1	62.4	38.0	75.6	54.1
ResNet-50 + EMS	36.5	63.1	71.7	45.8	57.5	59.3	47.9	32.3	69.9	63.5	37.3	76.0	55.1
DANN	39.8	58.0	68.1	48.6	57.0	59.9	46.9	38.4	68.8	63.2	47.7	75.9	56.0
DANN + EMS	40.3	60.4	70.1	49.6	57.5	60.5	47.9	38.3	69.1	63.7	47.7	76.8	56.8
ABAS-DANN	44.9	61.1	71.2	52.7	60.4	62.5	50.1	43.1	70.0	65.4	50.9	77.1	59.1
ALDA	46.4	68.6	74.6	57.6	67.0	69.4	57.2	46.3	75.6	69.2	53.3	80.8	63.8
ALDA + EMS	47.5	70.1	75.2	58.8	67.5	69.7	58.1	46.6	76.2	69.9	54.0	81.0	64.6
ABAS-ALDA	51.5	**71.7**	75.5	59.8	69.4	69.5	**59.8**	47.1	**77.7**	70.6	55.2	80.2	65.7

Table 2: Results of ABAS on Office-Home, using a ResNet-50 backbone. Note we were not able to reproduce the official ALDA numbers using the official code, + EMS: experiments run with our model selection strategy.

	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
ALDA [4]	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. Due to space constraints, ResNet50 and DANN experiments are in the supplementary materials.

References

[1] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[2] Fabio M Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2229–2238, 2019.

[3] Fabio Maria Carlucci, Paolo Russo, Tatiana Tommasi, and Barbara Caputo. Hallucinating agnostic images to generalize across domains. In *ICCV Workshops*, pages 3227–3234, 2019.

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

[5] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[6] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

[7] 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.

[8] 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.

[9] Yves Grandvalet, Yoshua Bengio, et al. Semi-supervised learning by entropy minimization. In *CAP*, pages 281–296, 2005.

[10] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations*, 2021.

[11] 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.

[12] Weixiang Hong, Zhenzhen Wang, Ming Yang, and Junsong Yuan. Conditional generative adversarial network for structured domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1335–1344, 2018.

[13] Yang Jiang, Cong Zhao, Zeyang Dou, and Lei Pang. Neural architecture refinement: A practical way for avoiding overfitting in nas. *arXiv preprint arXiv:1905.02341*, 2019.

[14] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

[15] Jingjing Li, Erpeng Chen, Zhengming Ding, Lei Zhu, Ke Lu, and Zi Huang. Cycle-consistent conditional adversarial transfer networks. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 747–755, 2019.

[16] Lisha Li, Kevin G Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *ICLR (Poster)*, 2017.

[17] Yichen Li and Xingchao Peng. Network architecture search for domain adaptation. *arXiv preprint arXiv:2008.05706*, 2020.

[18] Jian Liang, Ran He, Zhenan Sun, and Tieniu Tan. Distant supervised centroid shift: A simple and efficient approach to visual domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2975–2984, 2019.

[19] Hong Liu, Mingsheng Long, Jianmin Wang, and Michael Jordan. Transferable adversarial training: A general approach to adapting deep classifiers. In *International Conference on Machine Learning*, pages 4013–4022. PMLR, 2019.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[21] Mingsheng Long, ZHANGJIE CAO, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 1640–1650. Curran Associates, Inc., 2018.

[22] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*, pages 1640–1650, 2018.

[23] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *International conference on machine learning*, pages 2208–2217. PMLR, 2017.

[24] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709, 2018.

[25] Yawei Luo, Liang Zheng, Tao Guan, Junqing Yu, and Yi Yang. Taking a closer look at domain shift: Category-level adversaries for semantics consistent domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2507–2516, 2019.

[26] Masahiro Nomura and Yuta Saito. Multi-source unsupervised hyperparameter optimization. *arXiv preprint arXiv:2006.10600*, 2020.

[27] 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.

[28] Zhongyi Pei, Zhangjie Cao, Mingsheng Long, and Jianmin Wang. Multi-adversarial domain adaptation. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelli-*

gence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 3934–3941. AAAI Press, 2018.

[29] Pedro O Pinheiro. Unsupervised domain adaptation with similarity learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8004–8013, 2018.

[30] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

[31] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[32] Subhankar Roy, Aliaksandr Siarohin, Enver Sangineto, Samuel Rota Bulo, Nicu Sebe, and Elisa Ricci. Unsupervised domain adaptation using feature-whitening and consensus loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9471–9480, 2019.

[33] Binxin Ru, Pedro Esperanca, and Fabio Carlucci. Neural architecture generator optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

[34] Paolo Russo, Fabio M Carlucci, Tatiana Tommasi, and Barbara Caputo. From source to target and back: symmetric bi-directional adaptive gan. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8099–8108, 2018.

[35] 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.

[36] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3723–3732, 2018.

[37] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[38] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(5), 2007.

[39] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A Efros. Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*, 2019.

[40] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[41] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[42] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE international conference on computer vision*, pages 4068–4076, 2015.

[43] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[44] 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.

[45] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2019.

[46] 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.

[47] Yuan Wu, Diana Inkpen, and Ahmed El-Roby. Dual mixup regularized learning for adversarial domain adaptation. In *European Conference on Computer Vision*, pages 540–555. Springer, 2020.

[48] Minghao Xu, Jian Zhang, Bingbing Ni, Teng Li, Chengjie Wang, Qi Tian, and Wenjun Zhang. Adversarial domain adaptation with domain mixup. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6502–6509, 2020.

[49] Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly hard. In *International Conference on Learning Representations (ICLR)*, 2020.

[50] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[51] Kaichao You, Ximei Wang, Mingsheng Long, and Michael Jordan. Towards accurate model selection in deep unsupervised domain adaptation. In *International Conference on Machine Learning*, pages 7124–7133. PMLR, 2019.

[52] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

[53] Weichen Zhang, Wanli Ouyang, Wen Li, and Dong Xu. Collaborative and adversarial network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3801–3809, 2018.

[54] Sicheng Zhao, Xiangyu Yue, Shanghang Zhang, Bo Li, Han Zhao, Bichen Wu, Ravi Krishna, Joseph E Gonzalez, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, et al. A review of single-source deep unsupervised visual domain adaptation. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[55] Tianyi Zhou, Shengjie Wang, and Jeff Bilmes. Time-consistent self-supervision for semi-supervised learning. In *International Conference on Machine Learning*, pages 11523–11533. PMLR, 2020.

[56] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[57] Yang Zou, Zhiding Yu, BVK Vijaya Kumar, and Jinsong Wang. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In *Proceedings of the European conference on computer vision (ECCV)*, pages 289–305, 2018.

Adversarial Branch Architecture Search for Unsupervised Domain Adaptation Supplementary Materials

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:

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

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

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

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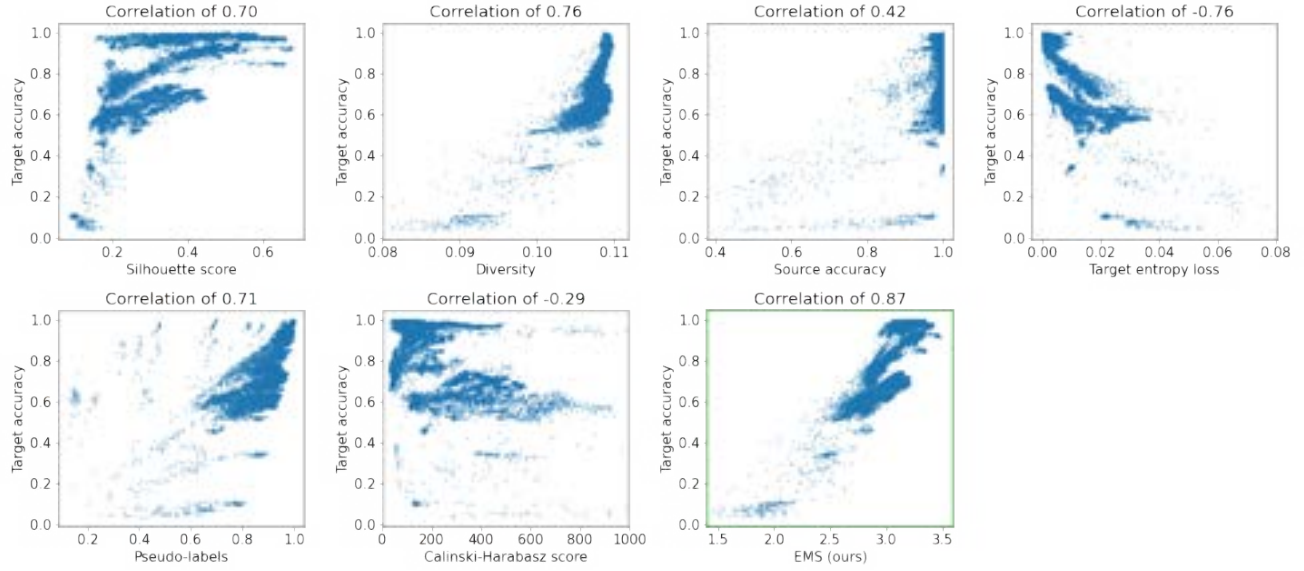
Figure 1: Correlation between the metrics introduced in section 3.3 and the target accuracy, as computed on Office31.
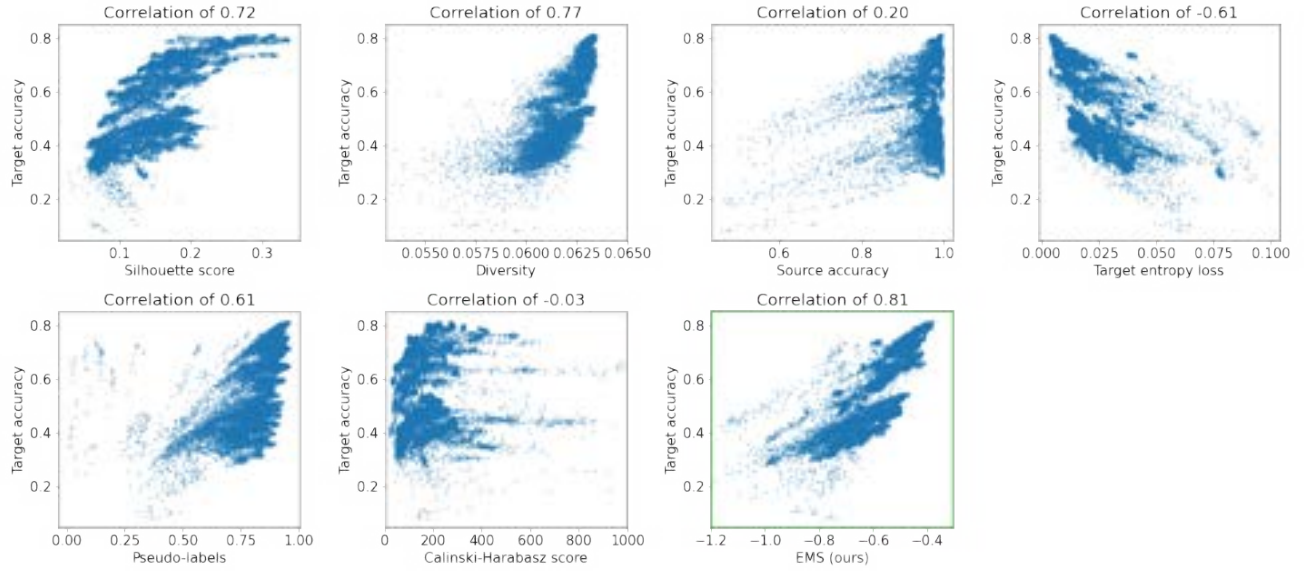

Figure 2: Correlation between the metrics introduced in section 3.3 and the target accuracy, as computed on Office-Home.

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.