CHALMERS
UNIVERSITY OF TECHNOLOGY

ECAP: EXTENSIVE CUT-AND-PASTE AUGMENTATION FOR UNSUPERVISED DOMAIN ADAPTIVE SEMANTIC SEGMENTATION

Citation for the original published paper (version of record):
Brorsson, E., Åkesson, K., Svensson, L. et al (2024). ECAP: EXTENSIVE CUT-AND-PASTE AUGMENTATION FOR UNSUPERVISED DOMAIN ADAPTIVE SEMANTIC SEGMENTATION. Proceedings - International Conference on Image Processing, ICIP: 610-616. http://dx.doi.org/10.1109/ICIP51287.2024.10647390

(article starts on next page)

ECAP: EXTENSIVE CUT-AND-PASTE AUGMENTATION FOR UNSUPERVISED DOMAIN ADAPTIVE SEMANTIC SEGMENTATION

Erik Brorsson[⋆†] *Knut Åkesson*[†] *Lennart Svensson*[†] *Kristofer Bengtsson*[⋆]

[⋆] Global Trucks Operations, Volvo Group, Gothenburg, Sweden
[†]Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

ABSTRACT

We consider unsupervised domain adaptation (UDA) for semantic segmentation in which the model is trained on a labeled source dataset and adapted to an unlabeled target dataset. Unfortunately, current self-training methods are susceptible to misclassified pseudo-labels resulting from erroneous predictions. Since certain classes are typically associated with less reliable predictions in UDA, reducing the impact of such pseudo-labels without skewing the training towards some classes is notoriously difficult. To this end, we propose an extensive cut-and-paste strategy (ECAP) to leverage reliable pseudo-labels through data augmentation. Specifically, ECAP maintains a memory bank of pseudo-labeled target samples throughout training and cut-and-pastes the most confident ones onto the current training batch. We implement ECAP on top of the recent method MIC and boost its performance on two synthetic-to-real domain adaptation benchmarks. Notably, MIC+ECAP reaches an unprecedented performance of 69.1 mIoU on the Synthia→Cityscapes benchmark. Our code is available at https://github.com/ErikBrorsson/ECAP.

Index Terms— Semantic Segmentation, Unsupervised Domain Adaptation, Pseudo-labeling, Data Augmentation, Self-training

1. INTRODUCTION

Unsupervised domain adaptation (UDA) is one of many approaches used for semantic segmentation that aims to relax the requirements on the availability of annotated training data. In UDA, an unlabeled *target* dataset sampled from the same distribution as the test dataset is available along with a labeled *source* dataset sampled from another distribution. The source dataset could constitute an already annotated dataset or a synthetic dataset for which annotations can easily be created. Due to the difference in distribution between source and target data, also known as the domain gap, a network trained on source data typically does not perform well on target data. Therefore, UDA methods use the unlabeled target data to adapt the model to the test data distribution, for example, through adversarial training [1, 2, 3, 4] or self-training [5, 6, 7, 8, 9, 10].

In recent years, self-training has dominated the field and is adopted by many recent works [11, 9, 12, 10]. A pivotal component of this framework is the DACS [6] data augmentation, which entails mixing a source and target image through a cut-and-paste operation. Although DACS augmentation is effective in bridging the domain gap, it does not handle the noise inherent to the pseudo-labels on which the model is trained. Most existing methods that address this problem attempt filtering the pseudo-labels based on predicted confidence scores [5, 7, 13, 14]. Unsurprisingly, the unconfident pseudo-labels often belong to *hard-to-adapt* classes, meaning that such approaches may impede the learning of these classes since the focus is shifted to *easy-to-adapt* classes. Multiple UDA methods facilitate learning hard-to-adapt classes by setting class-wise confidence thresholds when generating/filtering pseudo-labels [5, 7, 15] or using sampling schemes that favors these classes [11, 15, 16]. Nevertheless, maintaining a healthy balance between classes while focusing training on reliable pseudo-labels remains a challenge.

In this work, we take an entirely different approach to dealing with pseudo-label noise in self-training, which is based on cut-and-paste data augmentation. Specifically, we build a memory bank of pseudo-labeled target samples during training, and in each iteration, cut-and-paste confident samples from the memory bank onto the current training batch. By cut-and-pasting content from a large pool of images, our proposed method effectively makes use of the, typically scarce, confident pseudo-labels of the hard-to-adapt classes and shifts focus away from erroneous pseudo-labels during training. Our method, which we call ECAP: Extensive Cut-and-Paste, is to the best of our knowledge the first UDA method for semantic segmentation that aims to increase the proportion of correct pseudo-labels in each training image by means of cut-and-paste augmentation. Through comprehensive evaluation, ECAP is shown to increase the performance of multiple UDA methods based on self-training on the synthetic-to-real domain adaptation task. Notably, we reach new state-of-the-art performance on both the GTA→Cityscapes and Synthia→Cityscapes benchmarks by boosting the performance of the recent method MIC [10] by 0.3 and 1.8 mIoU respectively.

Our main contributions are:

1. We propose a data augmentation method for unsupervised domain adaptive semantic segmentation that is designed to counteract pseudo-label noise during self-training
2. We demonstrate that our approach increases the mIoU score of previous state-of-the-art on two synthetic-to-real domain adaptation benchmarks
3. We analyze the adverse effect of pseudo-label noise during self-training

2. RELATED WORK

Unsupervised Domain Adaptation (UDA) methods for semantic segmentation can broadly be categorized as either based on adversarial learning [1, 2, 3, 4] or self-training [5, 7, 6, 8, 9, 10]. In the former category, a discriminator network enforces domain invariance in the input through style-transferred images [1, 17] or in the feature/output space of the segmentation network [1, 17, 2, 3, 18]. In self-training, on the other hand, pseudo-labels are created for the unlabeled target data on which the model is then trained further. Multiple works also use an adaptation curriculum [19, 14], entropy minimization [3, 20, 7, 21] and more recently contrastive learning [12, 22].

Inspired by consistency regularization [23], many self-training methods enforce consistency between predictions on strongly augmented images with the pseudo-labels generated on weakly-augmented images. In particular, mixing the content of a source and target image is commonly used as the strong augmentation [6, 16, 24]. Furthermore, while pseudo-labels may be generated naively from the model's predictions [5, 7], many methods attempt increasing the quality of the pseudo-labels by e.g., using a mean teacher [6, 9], label prototypes [8, 25], or averaging predictions over multiple stochastic forward passes [24] or augmentations of the input [15].

The noise inherent to the pseudo-labels is often handled by either filtering the unreliable pseudo-labels by predicted confidence [5, 7, 13, 14], entropy [24] or uncertainty [26], or using a weighted loss function that assigns lower weight to unreliable pseudo-labels by e.g., model confidence [15], uncertainty [27] or depth estimation [28]. A problem with reducing the contribution of unreliable pseudo-labels is that the loss may become dominated by easy-to-adapt classes that are more reliable than the hard-to-adapt classes (these are often, although not necessarily, long-tail classes). To this end, many methods use class-wise confidence thresholds when generating/filtering pseudo-labels such that hard-to-adapt classes can be prioritized [5, 7, 15]. Others suggest using a focal loss to emphasize difficult samples [15] or data sampling schemes that favors such classes [11, 15, 16]. Different from all previous methods, ECAP aims to mitigate the issue of noisy pseudo-labels by increasing the proportion of correctly pseudo-labeled pixels in the training images by means of cut-and-paste data augmentation.

Cut-and-Paste Data Augmentation has been used both for image classification [29], object detection [30, 31, 32], semantic segmentation [33, 6, 21] and instance segmentation [34, 35, 32] in a variety of settings, including supervised learning [29, 31, 34, 35], semi-supervised learning [33, 34] and weakly-supervised learning [29, 32]. Notably, recent works [32] and [35] leverage generative models and foreground segmentation models to create large-scale datasets by cut-and-pasting multiple object instances into every training image. In UDA however, cut-and-paste augmentation has mainly been limited to methods that mix a single source and target image as a way of overcoming the domain gap. This idea, first proposed by DACS [6] has been used in many succeeding works [11, 9, 10, 12] and given rise to a number of variations [16, 24, 36]. Unlike all previous methods for UDA, ECAP is not restricted to images in a single batch but rather performs cut-and-paste data augmentation using a large pool of target images that are stored in a memory bank. Importantly, this allows ECAP to select the most suitable images for augmentation and training by considering the confidence of the associated pseudo-label.

3. PRELIMINARY

In this section, we establish the notation and provide relevant details for the self-training framework used in [6, 11, 9, 10] on top of which ECAP is implemented.

In self-training, a segmentation network f_θ (called the student) is trained on a set of N_S source domain images with associated labels $\{x^{S,k}, y^{S,k}\}_{k=1}^{N_S}$ and on a set of N_T unlabeled target domain images with associated pseudo-labels $\{x^{T,k}, \tilde{y}^{T,k}\}_{k=1}^{N_T}$. Dropping the index k, for ease of notation, the pseudo-label $\tilde{y}$ of an image x^T constitutes a one-hot vector $\tilde{y}_{ij}$ of length C at each pixel location (i, j), where C is the number of classes. The pseudo-label is created during training by the teacher network g_ϕ, which is an exponential moving average of the student's weights. Formally, the pseudo-label is defined by

$$\tilde{y}^T_{ijc} = [c = \text{argmax}_{c'} g_\phi(x^T)_{ijc'}], \tag{1}$$

where $[\cdot]$ denotes the Iverson bracket.

In every training iteration, a source and target image and their corresponding label and pseudo-label are mixed by a cut-and-paste operation wherein pixels given by a binary mask m are cut from the source sample and pasted onto the target sample. For convenience, we define such a mixing operation between two images x_1 and x_2 as

$$\alpha(x_1, x_2, m) := m \odot x_1 + (1 - m) \odot x_2, \tag{2}$$

where $\odot$ denotes element-wise multiplication. Given the operator α and binary mask m, the mixed image and corresponding label is constructed by $x^M = \alpha(x^S, x^T, m)$ and $y^M = \alpha(y^S, \tilde{y}^T, m)$.The binary mask m is created by randomly selecting half of the classes that are present in y^S and setting $m_{ij} = 1$ for all pixels included in the selections and $m_{ij} = 0$ otherwise.

Given the source and mixed image with corresponding labels as defined above, the weights θ of the student network f_θ are trained to minimize the loss

$$L(\theta) = \mathbb{E}[\mathcal{L}^S(y^S, f_\theta(x^S)) + \mathcal{L}^T(y^M, f_\theta(x^M), q^M)], \tag{3}$$

where $\mathcal{L}^S$ is the standard cross-entropy loss, $\mathcal{L}^T$ is a weighted cross-entropy loss, and the expectation is taken over data from the source and target datasets. Specifically, the second term is given by

$$\mathcal{L}^T(y^M, \hat{y}^M, q^M) = -\sum_{i=1}^{H}\sum_{j=1}^{W} q^M_{ij} \sum_{c=1}^{C} y^M_{ijc} log(\hat{y}^M_{ijc}), \tag{4}$$

where W and H are the width and height of the image x^M respectively, $\hat{y}^M$ denotes the predictions $f_\theta(x^M)$ of the student network, and q^M_{ij} is the weight associated with each pixel. In [6, 11, 9, 10], q^M_{ij} equals 1.0 for pixels originating from the source domain and otherwise equals the ratio of pixels in the target image for which the confidence of the pseudo-label exceeds a threshold τ, i.e.,

$$q^M_{ij} = \begin{cases} 1, & \text{if } m_{ij} = 1 \\ \frac{\sum_{i=1}^{H}\sum_{j=1}^{W}[(max_{c'} g_\phi(x^T)_{ijc'}) > \tau]}{H \cdot W}, & \text{otherwise.} \end{cases} \tag{5}$$

4. EXTENSIVE CUT-AND-PASTE (ECAP)

In this work, we propose to cut-and-paste confident pseudo-labeled content from multiple target domain images into the current training batch. Our method ECAP comprises three main components that are described in the following sections: (1) a memory bank containing pseudo-labeled target samples, (2) a sampler drawing samples associated with high confidence from the memory bank, and (3) an augmentation module that creates the augmented training images. We integrate ECAP with the self-training framework detailed in Section 3, but also recognize that ECAP may be applied in virtually any UDA method based on self-training. Figure 1 provides a schematic illustration of our method.

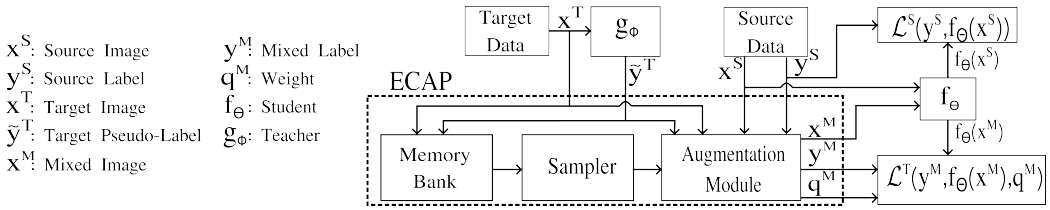

Fig. 1. Schematic illustration of ECAP, constituting a memory bank, a sampler, and an augmentation module, integrated with the self-training framework. The input to ECAP is a source and target image, x^S and x^T, along with the corresponding label y^S and pseudo-label $\tilde{y}^T$, which is produced by the teacher g_Φ. This input is processed by the augmentation module together with samples from the memory bank to generate a mixed image x^M, an associated label y^M, and a weight q^M. Simultaneously, x^T and $\tilde{y}^T$ are added to the memory bank for future use. The student network f_θ processes both the source image x^S and the mixed image x^M and is supervised by the loss $\mathcal{L}^S(y^S, f_\theta(x^S)) + \mathcal{L}^T(y^M, f_\theta(x^M), q^M)$ during training.

4.1. Memory Bank

During training, a memory bank B_c is constructed for each class c in the dataset. Each memory bank B_c consists of a set of $|B_c|$ images, pseudo-labels and confidence scores $\{(x^{B_c,l}, y^{B_c,l}, q^{B_c,l})\}_{l=1}^{|B_c|}$. Again, dropping the index l for ease of notation, x^{B_c} and y^{B_c} are constructed by applying a binary mask m_c to a target image x^T and pseudo-label $\tilde{y}^T$ according to $(x^{B_c}, y^{B_c}) = (x^T \odot m_c, \tilde{y}^T \odot m_c)$. The binary mask m_c takes the value 1.0 at each pixel location (i, j) for which $\tilde{y}_{ij}^T$ is of class c, and otherwise m_c takes the value 0. Furthermore, the confidence score q^{B_c} is defined as the average confidence of the teacher's predictions corresponding to pseudo-label of class c for the target image x^T according to

$$q^{B_c} = \frac{\sum_{i=1}^{W}\sum_{j=1}^{H} g_\phi(x^T)_{ijc}[c = \text{argmax}_{c'}(g_\phi(x^T)_{ijc'})]}{\sum_{i=1}^{W}\sum_{j=1}^{H}[c = \text{argmax}_{c'}(g_\phi(x^T)_{ijc'})]}. \quad (6)$$

Note that x^{B_c}, y^{B_c} and q^{B_c} are computed for every class c present in $\tilde{y}^T$, which implies that (different parts of) a single target image x^T and the corresponding pseudo-label $\tilde{y}^T$ are often added to multiple memory banks. To avoid storing duplicates of certain target samples as training spans multiple epochs, we allow at most one sample in each memory bank B_c per target image.

4.2. Sampler

In each training iteration, a set of samples $\{S^{B_c,l}\}_{c,l} = \{(x^{B_c,l}, y^{B_c,l})\}_{c,l}$ is obtained by first selecting a set of classes and then drawing a sample from the memory bank of each selected class (we use a slight abuse of notation here as the index pair c, l takes on values that depend on the outcome of the sampling). Formally, we define a set of random variables $\{\mathbf{r_c} \sim \text{Bern}(p_{r_c})\}_{c=1}^{C}$ of which a set of observations $\{r_c\}_{c=1}^{C}$ is obtained in each training iteration. For every class c, one sample is then drawn from memory bank B_c if $r_c = 1$. Since samples of high quality are preferred, we use the confidence value q^{B_c}, defined in Equation 6, when sampling from each memory bank. Specifically, we propose uniform sampling from the top n^{B_c} most confident samples in each memory bank B_c, where n^{B_c} is a hyperparameter. Consequently, if we sort the confidence scores of B_c in descending order and let $q(n^{B_c})$ be the score at position n^{B_c}, the sampling probability $p^{B_c,l}$ of each sample in B_c can be defined as

$$p^{B_c,l} = \begin{cases} 1/n^{B_c}, & \text{if } q^{B_c,l} \geq q(n^{B_c}) \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

Furthermore, we argue that the sampling probability p_{r_c} should increase as training progresses and the quality of the memory bank increases. In this work, we let p_{r_c} be equal across all classes and define $p_{r_c} = n_0\sigma(\frac{MEC-\beta}{\gamma})$, where $n_0 \in [0, 1]$ is a constant, MEC is the mean expected confidence of a sample drawn uniformly from the set of memory banks $\{B\}_c$, σ is the *sigmoid* function, and β and γ are two hyperparameters. With these design choices, p_{r_c} is given by

$$p_{r_c} = n_0\sigma(\frac{\frac{1}{C}\sum_{c=1}^{C}\sum_{l=1}^{|B_c|} q^{B_c,l}p^{B_c,l} - \beta}{\gamma}), \quad (8)$$

where the index l runs over every sample (with associated confidence value $q^{B_c,l}$ and sampling probability $p^{B_c,l}$) in memory bank B_c. Note that β can be thought of as a typical value of MEC at which ECAP sampling comes online, while γ relates to how quickly sampling comes online as MEC approaches β.

4.3. Augmentation Module

The set of samples $\{S^{B_c,l}\}_{c,l}$ obtained by the sampler in Section 4.2 is used to create a composite image x^B and corresponding pseudo-label y^B. This is done by initializing x^B and y^B as blank canvases, iterating over the samples in $\{S^{B_c,l}\}_{c,l}$, and in each iteration cut-and-pasting the content of $S^{B_c,l}$ corresponding to class c onto the canvases. Before pasting, however, the sample is subject to a series of transformations, including random scaling, translation and horizontal flipping. The construction of x^B and y^B is described in Algorithm 1, where α is the mixing operator defined in Equation 2.

Algorithm 1 Creating composite image

Initialize x^B and y^B as blank canvases.
$\{S^{B_c,l}\}_{c,l} \leftarrow$ a set of samples provided by the Sampler.
Shuffle $\{S^{B_c,l}\}_{c,l}$.
for $(x^{B_c,l}, y^{B_c,l})$ in $\{S^{B_c,l}\}_{c,l}$ **do**
  $T(x^{B_c,l}), T(y^{B_c,l}) \leftarrow$ Applying a set of random transformations to the sample.
  $m_c \leftarrow T(y^{B_c,l}) = c$
  $x^B \leftarrow \alpha(T(x^{B_c,l}), x^B, m_c)$
  $y^B \leftarrow \alpha(T(y^{B_c,l}), y^B, m_c)$
end for
return x^B, y^B

The composite image x^B and corresponding pseudo-label y^B are then used to alter the mixed image x^M and corresponding

pseudo-label y^M, which are used in self-training as detailed in Section 3. Specifically, we paste x^B (and y^B) onto the source image x^S (and label y^S) *prior* to DACS mixing. Formally, if we let DACS$(x^S, x^T, y^S, \tilde{y}^T)$ denote the algorithm detailed in Section 3 to generate a mixed sample and associated weight from a source and target sample, we let

$$\begin{aligned}&(x^M, y^M, q^M) = \\ &\text{DACS}(\alpha(x^B, x^S, m^B), x^T, \alpha(y^B, y^S, m^B), \tilde{y}^T),\end{aligned} \tag{9}$$

where again α denotes the mixing operator defined in Equation 2 and m^B is a binary mask corresponding to the parts of x^B that has been populated. Note that m^B can be trivially computed by a small addition to Algorithm 1, although we exclude it for simplicity. Also note that the binary mask used internally in the DACS algorithm to construct x^M, y^M, and q^M is now derived by selecting half of the classes in $\alpha(y^B, y^S, m^B)$, which means that pixels in x^M originating from the composite image x^B will be associated with a weight of 1.0. This design choice is made since the pseudo-label of x^B is expected to be of high quality.

5. EXPERIMENTS

5.1. Implementation Details

Datasets: We study ECAP in the setting of synthetic-to-real domain adaptation on the popular benchmarks GTA→Cityscapes and Synthia→Cityscapes. The GTA [37] and Synthia [38] datasets consist of 24,966 and 9,000 simulated training images respectively along with accompanying labels. The Cityscapes [39] training dataset consists of 2,975 images, which are used as the unlabeled target dataset. In line with previous works, we evaluate the performance on the Cityscapes validation set of 500 images. Additionally, we study day-to-nighttime and clear-to-adverse-weather domain adaptation on the benchmarks Cityscapes→DarkZurich and Cityscapes→ACDC respectively. The DarkZurich [40] dataset is captured during nighttime and consist of 2,416 training and 151 test images, while the ACDC [41] dataset is captured in adverse weather and comprises 1,600 training and 2,000 test images. The Cityscapes training dataset is used as the labeled source dataset for these benchmarks.

Training: In this work, MIC [10] is used as a baseline on top of which ECAP is implemented. MIC is based on the self-training framework detailed in Section 3 with the addition of Rare Class Sampling [11], ImageNet Feature Distance [11] and multi-resolution fusion [9], as well as Masked Image Consistency [10]. We use the exact training parameters of MIC [10] and refer the reader to [10] for details. The only difference when applying ECAP is that the mixed training sample is created according to Equation 9.

For simplicity, we let n^{B_c} in Equation 7 be equal for all classes and set $\gamma = 0.005$ in Equation 8 in all our experiments. In the augmentation module of ECAP, we apply random scaling by a factor r_s sampled uniformly on the interval $[0.1, 1.0]$, as well as random translation and horizontal flipping. In the experiments on Synthia, we disable sampling from the ECAP memory banks corresponding to the classes *Terrain*, *Truck* and *Train* since these don't exist in the Synthia dataset. Furthermore, while MIC [10] refrains from training on pseudo-labels in the region of the Cityscapes images corresponding to the ego-vehicle hood as well as on the image borders, we find this detrimental in the case of Synthia→Cityscapes. Therefore, we disable this feature and instead train on the entire Cityscapes image for this benchmark. This is elaborated further in the supplementary material available at https://sigport.org/documents/ecap-supplementary.

5.2. Comparison with State-of-the-Art

We compare ECAP with existing UDA methods on four popular benchmarks in Table 1. The reported mean and standard deviation of each experiment are computed from three runs with different random seeds. We use the following hyperparameters of ECAP: $n_0 = 1.0$, $\beta = 0.95$, $n_c^{B_c} = 40$ for GTA→Cityscapes, $n_0 = 1.0$, $\beta = 0.80$, $n_c^{B_c} = 30$ for Synthia→Cityscapes, $n_0 = 0.53$, $\beta = 0.98$, $n_c^{B_c} = 30$ for Cityscapes→DarkZurich, and $n_0 = 1.0$, $\beta = 0.90$, $n_c^{B_c} = 50$ for Cityscapes→ACDC.

On GTA→Cityscapes, ECAP gives a modest boost of 0.3 mIoU to MIC. On Synthia→Cityscapes, we report the results of MIC as well as MIC†, which is a variant of MIC that trains on pseudo-labels in the entire target image. We find that MIC† outperforms MIC by 0.9 mIoU, showing that it is beneficial to train on the entire image for this benchmark. When additionally applying ECAP, performance is boosted further by another 0.9 mIoU, reaching an unprecedented performance of 69.1 mIoU.

On the other hand, ECAP degrades the performance of MIC on Cityscapes→DarkZurich and Cityscapes→ACDC due to large drops in IoU for certain classes, such as road and sidewalk, although some classes benefit from ECAP, especially wall and fence in Cityscapes→ACDC. We hypothesize that ECAP is not as suitable for these domain adaptation benchmarks since the appearance of objects is a less discriminative factor for images with poor visibility. Instead, the context and prior knowledge about the scene becomes increasingly important to correctly segment images under such conditions. Since ECAP is based on aggressive data augmentation, it may hamper learning of context (e.g., the sky typically appears above the road in the images), which may be a significant issue in low visibility conditions. Qualitative results and an extended analysis is provided in the supplement.

5.3. ECAP on Other Methods

We also implement ECAP on various prior art models on the GTA→Cityscapes benchmark to understand how well ECAP generalizes across different methods. The results for each model without ECAP is taken from [10], while the results with ECAP are computed from three experiments on different random seeds. In Table 2, it can be seen that ECAP gives a substantial performance boost to all investigated methods, showing that it's not designed specifically for MIC. Noteably, both convolutional neural networks and transformer architectures benefit from ECAP. Furthermore, it is apparent that less capable models benefit more from ECAP than current state-of-the-art. This is expected since improving state-of-the-art becomes increasingly difficult as performance saturates.

5.4. In-Depth Analysis of ECAP

This section provides an in-depth analysis of ECAP and specifically addresses the issue of pseudo-label noise, which has been the main driver of our proposed method. To save time, experiments are done on a single random seed and conducted with DAFormer since it has a substantially faster training time than MIC. In Table 3, the results from training four different variants of DAFormer on GTA→Cityscapes are presented. Two of the columns correspond to standard DAFormer and DAFormer+ECAP (with the same settings as in Section 5.3). On the other hand, DAFormer (denoise) uses the target domain labels to set the pixel-level weights to zero for any incorrect pseudo-labels, thereby eliminating such pseudo-labels' contribution to the training (recall that a weighted cross-entropy loss function is used). Conversely, DAFormer (oracle) directly replaces

Table 1. Semantic segmentation performance (IoU in %) on four different UDA benchmarks.

Method	Road	Sidew.	Build.	Wall	Fence	Pole	Tr.Light	Tr.Sign	Veget.	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	M.bike	Bike	mIoU
GTA→Cityscapes (Val.)																				
DACS [6]	89.9	39.7	87.9	30.7	39.5	38.5	46.4	52.8	88.0	44.0	88.8	67.2	35.8	84.5	45.7	50.2	0.0	27.3	34.0	52.1
HRDA [9]	96.4	74.4	91.0	61.6	51.5	57.1	63.9	69.3	91.3	48.4	94.2	79.0	52.9	93.9	84.1	85.7	75.9	63.9	67.5	73.8
PiPa [12]	96.8	76.3	91.6	**63.0**	57.7	60.0	65.4	**72.6**	**91.7**	51.8	**94.8**	79.7	56.4	94.4	**85.9**	88.4	78.9	63.5	67.2	75.6
MIC [10]	**97.4**	80.1	**91.7**	61.2	56.9	59.7	**66.0**	71.3	**91.7**	51.4	94.3	79.8	56.1	**94.6**	85.4	90.3	80.4	64.5	**68.5**	75.9
MIC+ECAP	**97.4**	**80.3**	91.6	60.4	**58.2**	**60.9**	65.6	71.8	**91.7**	**52.8**	93.9	**80.6**	**57.2**	94.4	85.2	**91.1**	**82.1**	**65.2**	67.8	**76.2**
	±0.1	**±0.4**	±0.0	±1.9	**±0.6**	**±0.7**	±0.6	±1.0	**±0.1**	**±0.5**	±0.2	**±0.5**	**±1.4**	±0.1	±2.2	**±0.1**	**±1.4**	**±0.4**	±0.7	**±0.1**
Synthia→Cityscapes (Val.)																				
DACS [6]	80.6	25.1	81.9	21.5	2.9	37.2	22.7	24.0	83.7		90.8	67.6	38.3	82.9		38.9		28.5	47.6	48.3
HRDA [9]	85.2	47.7	88.8	49.5	4.8	57.2	65.7	60.9	85.3		92.9	79.4	52.8	89.0		64.7		63.9	64.9	65.8
PiPa [12]	88.6	50.1	**90.0**	**53.8**	7.7	58.1	67.2	63.1	88.5		94.5	79.7	57.6	90.8		70.2		65.1	**66.9**	68.2
MIC [10]	86.6	50.5	89.3	47.9	7.8	**59.4**	66.7	**63.4**	87.1		94.6	81.0	**58.9**	90.1		61.9		**67.1**	64.3	67.3
MIC†	**91.0**	**55.7**	89.9	50.4	**8.4**	58.8	66.7	62.9	**89.2**		94.6	81.2	57.6	90.6		65.3		66.0	63.2	68.2
	±0.9	**±2.0**	±0.1	±1.7	**±0.1**	±0.4	±0.2	±0.4	**±1.2**		±0.1	±0.1	±0.5	±0.3		±5.7		±0.7	±1.7	±0.2
MIC†+ECAP	90.8	55.2	**90.0**	50.7	8.2	59.3	**68.3**	63.0	89.0		**94.8**	**81.8**	58.6	**90.9**		**71.4**		**67.1**	65.9	**69.1**
	±0.8	±1.6	**±0.1**	±3.7	±0.6	±0.2	**±0.3**	±0.2	±0.3		**±0.2**	**±0.3**	±0.3	**±0.0**		**±0.4**		**±1.3**	±0.5	**±0.3**
Cityscapes→Dark Zurich (Test)																				
DAFormer [11]	93.5	65.5	73.3	39.4	19.2	53.3	44.1	44.0	59.5	34.5	66.6	53.4	52.7	82.1	52.7	9.5	89.3	50.5	38.5	53.8
HRDA [9]	90.4	56.3	72.0	39.5	19.5	57.8	**52.7**	43.1	59.3	29.1	70.5	60.0	58.6	**84.0**	**75.5**	11.2	90.5	51.6	40.9	55.9
MIC [10]	**94.8**	**75.0**	**84.0**	**55.1**	**28.4**	62.0	35.5	**52.6**	59.2	**46.8**	70.0	**65.2**	**61.7**	82.1	64.2	**18.5**	91.3	**52.6**	44.0	60.2
MIC+ECAP	90.6	55.8	82.2	53.4	25.0	**62.1**	38.2	51.4	**63.5**	43.1	**73.3**	63.9	59.1	83.1	62.2	16.2	**91.8**	47.4	**47.0**	58.4
	±2.3	±9.5	±1.4	±4.4	±3.1	**±1.0**	±8.9	±2.2	**±5.1**	±2.3	**±5.7**	±0.3	±0.4	±0.6	±0.8	±2.7	**±0.3**	±6.6	**±0.7**	±1.2
Cityscapes→ACDC (Test)																				
DAFormer [11]	58.4	51.3	84.0	42.7	35.1	50.7	30.0	57.0	74.8	52.8	51.3	58.3	32.6	82.7	58.3	54.9	82.4	44.1	50.7	55.4
HRDA [9]	88.3	57.9	88.1	55.2	36.7	56.3	**62.9**	65.3	74.2	57.7	85.9	68.8	45.7	88.5	**76.4**	82.4	87.7	52.7	60.4	68.0
MIC [10]	**90.8**	**67.1**	**89.2**	54.5	40.5	**57.2**	62.0	**68.4**	76.3	61.8	**87.0**	**71.3**	**49.4**	**89.7**	75.7	**86.8**	**89.1**	**56.9**	**63.0**	**70.4**
MIC+ECAP	69.5	56.9	**89.2**	**57.5**	**43.8**	56.4	49.6	67.2	**77.0**	**62.8**	66.6	**71.3**	43.2	89.4	73.2	68.0	88.7	56.8	62.3	65.8
	±9.4	±5.1	**±0.2**	**±1.0**	**±1.4**	±0.4	±13.8	±0.6	**±1.0**	**±1.1**	±12.1	**±0.4**	±0.4	±0.4	±1.7	±3.8	±0.5	±0.6	±1.0	±2.1

Table 2. Performance of different UDA methods without and with ECAP (mIoU in %).

Network	UDA Method	w/o ECAP	w/ ECAP	diff
DeepLabV2	DACS	53.9	58.3	+4.4
DeepLabV2	DAFormer	56.0	61.2	+5.2
DeepLabV2	HRDA	63.0	65.6	+2.6
DeepLabV2	MIC	64.2	66.3	+2.1
DAFormer	DAFormer	68.3	69.1	+0.8
DAFormer	HRDA	73.8	75.0	+1.2
DAFormer	MIC	75.9	76.2	+0.3

all pseudo-labels with the corresponding labels, serving as an upper bound for DAFormer's performance. Along with mIoU of the four models, we also present the *target accuracy*, which is the proportion of correct pseudo-labels, as well as the *target loss noise ratio*, which is the proportion of the target loss that is derived from incorrect pseudo-labels. The target loss refers to the part of the loss that is derived from target domain pixels, which may originate from either the sampled target image or the ECAP memory bank. The target accuracy and target loss noise ratio are computed on the mixed training images and are averaged over the final 50 iterations of training.

In Table 3, the performance of DAFormer (oracle) is 5.8 mIoU points higher than DAFormer. Moreover, DAFormer (denoise) significantly narrows this gap by 4.0 mIoU points, demonstrating that shifting focus towards correct pseudo-labels during training is an effective strategy. Furthermore, ECAP increases the target accuracy of DAFormer to a level on par with DAFormer (denoise) and additionally lowers the target loss noise ratio of DAFormer. This indicates that augmenting the training examples with ECAP increases the proportion of correctly pseudo-labeled content and as a result, the remaining erroneous pseudo-labels make up a smaller proportion of the loss value. Importantly, ECAP increases the mIoU of DAFormer significantly, although not as much as DAFormer (denoise) and (oracle) which both have access to the target domain labels.

Table 3. Performance (in %) of four variants of DAFormer.

	DAFormer	DAFormer (denoise)	DAFormer (oracle)	ECAP (DAFormer)
mIoU	68.0	72.0	73.8	68.7
Target accuracy	87.8	89.3	100.0	89.6
Target loss noise ratio	37.9	0.0	0.0	33.3

Table 4 further shows the target accuracy of DAFormer+ECAP (from Table 3) split over different classes and pixels originating from the ECAP memory bank and the sampled target image respectively. It is evident that the content from the memory bank typically is associated with more accurate pseudo-labels. This is especially true for *thing classes* such as train, motorbike and bike, while not as apparent for *stuff classes* such as road, sidewalk and building.

5.5. Hyperparameter Sensitivity Analysis

This section provides a sensitivity analysis of the hyperparameters of ECAP. To save time, we perform experiments on a single seed on GTA→Cityscapes using DAFormer+ECAP ($n_0 = 1.0, \beta =$

Table 4. Accuracy (in %) of pseudo-labels for pixels originating from the ECAP memory bank and the sampled target image.

	Road	Sidew.	Build.	...	Train	M.bike	Bike
Memory bank	94.3	67.3	89.9		96.1	80.7	76.3
Target image	91.9	60.5	90.4		55.0	37.5	57.4

$0.93, n_c^{B_c} = 30$) as a baseline and change the hyperparameters one at a time. Table 5 reports the performance and deviation from this baseline in terms of mIoU under a number of different settings.

Table 5. Performance of ECAP (mIoU %) under different hyperparameter settings. The Δ-row displays the deviation from the ECAP baseline and ECAP$^-$ indicates the removal of random scaling, translation and flipping in the augmentation module.

	n_0		β		$n_c^{B_c}$			ECAP$^-$	DAFormer	ECAP
	0.053	0.53	0.75	0.97	5	50	100			
Δ	-0.6	+0.2	$\pm$0.0	-0.8	-1.0	+0.2	-0.7	$\pm$0.0	-1.3	$\pm$0.0
mIoU	68.5	69.3	69.1	68.3	68.1	69.3	68.4	69.1	67.8	69.1

ECAP intensity n_0: Since ECAP is reduced to DAFormer when $n_0 = 0$ it is expected that the performance approaches that of DAFormer as the value of n_0 decreases. In Table 5, it can be seen that $n_0 = 0.053$ achieves lower performance than the ECAP baseline, although still superior to DAFormer. However, $n_0 = 0.53$ performs slightly better than the ECAP baseline, which indicates that $n_0 = 1.0$ (which out of convenience was used in most experiments of our paper) may not be optimal.

Sampling schedule β: We analyze the impact of changing the value of β which determines when ECAP sampling comes online during training. In practice $\beta = 0.75$, $\beta = 0.9$ (baseline) and $\beta = 0.97$ implies that ECAP comes online roughly at iteration 3k, 8k and 20k respectively. In Table 5, it can be seen that ECAP performs well even when starting sampling very early in training, indicating that doing ECAP augmentation with initially less confident pseudo-labels doesn't impede learning. On the other hand, an excessively large value of β results in performance similar to DAFormer, which is expected since ECAP is reduced to DAFormer as β approaches 1.0.

Effective memory bank size $n_c^{B_c}$: In Table 5 it can be seen that $n_c^{B_c} = 50$ performs slightly better than the baseline $n_c^{B_c} = 30$, while $n_c^{B_c} = 5$ and $n_c^{B_c} = 100$ perform significantly worse than the baseline, although still advantageous in comparison to DAFormer. We hypothesize that using an unnecessarily small memory bank is suboptimal since it implies less diversity of the ECAP samples. Conversely, an excessively large memory bank implies that less confident samples are included in the memory bank, which may lower ECAP's effectiveness in reducing pseudo-label noise.

Transformations in the augmentation module: In Table 5, ECAP$^-$ denotes the removal of the random scaling, translation, and horizontal flipping included in the augmentation module of ECAP. We note that removing these components doesn't effect the performance in this experiment and is not essential to the functioning of ECAP.

6. CONCLUSIONS

In this paper, we presented ECAP, a data augmentation method designed to reduce the adverse effect of erroneous pseudo-labels for unsupervised domain adaptive semantic segmentation. By cut-and-pasting confident pseudo-labeled target samples from a memory bank, ECAP benefits training by shifting focus away from erroneous pseudo-labels. Through comprehensive experiments, we demonstrate the effectiveness of our approach on synthetic-to-real domain adaptation. Notably, we boost the performance of the recent method MIC with 0.3 mIoU on GTA→Cityscapes and 1.8 mIoU on Synthia→Cityscapes, setting new state-of-the-art performance in both cases. Our experiments on day-to-nighttime and clear-to-adverse-weather domain adaptation benchmarks additionally highlights a limitation of ECAP. Namely that ECAP may hamper the learning of context information and generate predictions with less bias and higher variance following training with aggressive data augmentation. Therefore, we find ECAP less suitable for adaptation to domains with e.g., poor visibility, where context information and a strong bias is pivotal for making accurate predictions. Thanks to the demonstrated benefits of ECAP on synthetic-to-real UDA, we hope that ECAP can be part of future UDA methods to further push the state-of-the-art on this important problem.

Acknowledgment. This work was supported by AB Volvo and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The experiments were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) at Chalmers Centre for Computational Science and Engineering (C3SE) partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973.

7. REFERENCES

[1] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, "CyCADA: Cycle-consistent adversarial domain adaptation," in *ICML*, vol. 80, 10–15 Jul 2018, pp. 1989–1998.

[2] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker, "Learning to adapt structured output space for semantic segmentation," in *CVPR*, June 2018, pp. 7472–7481.

[3] T.-H. Vu, H. Jain, M. Bucher, M. Cord, and P. Perez, "ADVENT: Adversarial entropy minimization for domain adaptation in semantic segmentation," in *CVPR*, June 2019, pp. 2517–2526.

[4] Y.-H. Tsai, K. Sohn, S. Schulter, and M. Chandraker, "Domain adaptation for structured output via discriminative patch representations," in *ICCV*, October 2019, pp. 1456–1465.

[5] Y. Zou, Z. Yu, B. V. Kumar, and J. Wang, "Unsupervised domain adaptation for semantic segmentation via class-balanced self-training," in *ECCV*, September 2018, pp. 289–305.

[6] W. Tranheden, V. Olsson, J. Pinto, and L. Svensson, "DACS: Domain adaptation via cross-domain mixed sampling," in *WACV*, January 2021, pp. 1379–1389.

[7] K. Mei, C. Zhu, J. Zou, and S. Zhang, "Instance adaptive self-training for unsupervised domain adaptation," in *ECCV*. Springer, 2020, pp. 415–430.

[8] P. Zhang, B. Zhang, T. Zhang, D. Chen, Y. Wang, and F. Wen, "Prototypical pseudo label denoising and target structure learning for domain adaptive semantic segmentation," in *CVPR*, June 2021, pp. 12 414–12 424.

[9] L. Hoyer, D. Dai, and L. Van Gool, "HRDA: Context-aware high-resolution domain-adaptive semantic segmentation," in *ECCV*, 2022, pp. 372–391.

[10] L. Hoyer, D. Dai, H. Wang, and L. Van Gool, "MIC: Masked image consistency for context-enhanced domain adaptation," in *CVPR*, June 2023, pp. 11 721–11 732.

[11] L. Hoyer, D. Dai, and L. Van Gool, "DAFormer: Improving network architectures and training strategies for domain-adaptive semantic segmentation," in *CVPR*, 2022, pp. 9924–9935.

[12] M. Chen, Z. Zheng, Y. Yang, and T.-S. Chua, "PiPa: Pixel- and patch-wise self-supervised learning for domain adaptative semantic segmentation," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, p. 1905–1914.

[13] Y. Zou, Z. Yu, X. Liu, B. V. Kumar, and J. Wang, "Confidence regularized self-training," in *ICCV*, October 2019, pp. 5982–5991.

[14] Q. Lian, F. Lv, L. Duan, and B. Gong, "Constructing self-motivated pyramid curriculums for cross-domain semantic segmentation: A non-adversarial approach," in *ICCV*, October 2019, pp. 6758–6767.

[15] N. Araslanov and S. Roth, "Self-supervised augmentation consistency for adapting semantic segmentation," in *CVPR*, June 2021, pp. 15 384–15 394.

[16] L. Gao, J. Zhang, L. Zhang, and D. Tao, "DSP: Dual soft-paste for unsupervised domain adaptive semantic segmentation," in *Proceedings of the 29th ACM International Conference on Multimedia*. ACM, oct 2021, p. 2825–2833.

[17] Y. Li, L. Yuan, and N. Vasconcelos, "Bidirectional learning for domain adaptation of semantic segmentation," in *CVPR*, June 2019, pp. 6936–6945.

[18] K. Saito, K. Watanabe, Y. Ushiku, and T. Harada, "Maximum classifier discrepancy for unsupervised domain adaptation," in *CVPR*, June 2018, pp. 3723–3732.

[19] Y. Zhang, P. David, and B. Gong, "Curriculum domain adaptation for semantic segmentation of urban scenes," in *ICCV*. IEEE, oct 2017, pp. 2020–2030.

[20] M. Chen, H. Xue, and D. Cai, "Domain adaptation for semantic segmentation with maximum squares loss," in *ICCV*, 2019, pp. 2090–2099.

[21] C. Zhu, K. Liu, W. Tang, K. Mei, J. Zou, and T. Huang, "Hard-aware instance adaptive self-training for unsupervised cross-domain semantic segmentation," *arXiv preprint arXiv:2302.06992*, 2023.

[22] J. Huang, D. Guan, A. Xiao, S. Lu, and L. Shao, "Category contrast for unsupervised domain adaptation in visual tasks," in *CVPR*, June 2022, pp. 1203–1214.

[23] K. Sohn, D. Berthelot, N. Carlini, Z. Zhang, H. Zhang, C. A. Raffel, E. D. Cubuk, A. Kurakin, and C.-L. Li, "FixMatch: Simplifying semi-supervised learning with consistency and confidence," in *NeurIPS*, vol. 33, 2020, pp. 596–608.

[24] Q. Zhou, Z. Feng, Q. Gu, J. Pang, G. Cheng, X. Lu, J. Shi, and L. Ma, "Context-aware mixup for domain adaptive semantic segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 2, pp. 804–817, 2023.

[25] Q. ZHANG, J. Zhang, W. Liu, and D. Tao, "Category anchor-guided unsupervised domain adaptation for semantic segmentation," in *NeurIPS*, vol. 32, 2019, pp. 435–445.

[26] Q. Zhou, Z. Feng, Q. Gu, G. Cheng, X. Lu, J. Shi, and L. Ma, "Uncertainty-aware consistency regularization for cross-domain semantic segmentation," *Computer Vision and Image Understanding*, vol. 221, no. 103448, 2022.

[27] Z. Zheng and Y. Yang, "Rectifying pseudo label learning via uncertainty estimation for domain adaptive semantic segmentation," *IJCV*, vol. 129, no. 4, pp. 1106–1120, 2021.

[28] Q. Wang, D. Dai, L. Hoyer, L. Van Gool, and O. Fink, "Domain adaptive semantic segmentation with self-supervised depth estimation," in *ICCV*, October 2021, pp. 8515–8525.

[29] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "CutMix: Regularization strategy to train strong classifiers with localizable features," in *ICCV*, October 2019, pp. 6023–6032.

[30] D. Dwibedi, I. Misra, and M. Hebert, "Cut, paste and learn: Surprisingly easy synthesis for instance detection," in *ICCV*, 2017, pp. 1301–1310.

[31] N. Dvornik, J. Mairal, and C. Schmid, "Modeling visual context is key to augmenting object detection datasets," in *ECCV*, 2018, pp. 364–380.

[32] Y. Ge, J. Xu, B. N. Zhao, L. Itti, and V. Vineet, "EM-Paste: EM-guided cut-paste with DALL-E augmentation for image-level weakly supervised instance segmentation," *arXiv preprint arXiv:2212.07629*, 2022.

[33] V. Olsson, W. Tranheden, J. Pinto, and L. Svensson, "ClassMix: Segmentation-based data augmentation for semi-supervised learning," in *WACV*, January 2021, pp. 1369–1378.

[34] G. Ghiasi, Y. Cui, A. Srinivas, R. Qian, T.-Y. Lin, E. D. Cubuk, Q. V. Le, and B. Zoph, "Simple copy-paste is a strong data augmentation method for instance segmentation," in *CVPR*, 2021, pp. 2918–2928.

[35] H. Zhao, D. Sheng, J. Bao, D. Chen, D. Chen, F. Wen, L. Yuan, C. Liu, W. Zhou, Q. Chu, W. Zhang, and N. Yu, "X-paste: Revisiting scalable copy-paste for instance segmentation using CLIP and StableDiffusion," in *ICML*, vol. 202, 23–29 Jul 2023, pp. 42 098–42 109.

[36] Z. Chen, Z. Ding, J. M. Gregory, and L. Liu, "IDA: Informed domain adaptive semantic segmentation," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 90–97.

[37] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *ECCV*, vol. 9906, 2016, pp. 102–118.

[38] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *CVPR*, June 2016, pp. 3234–3243.

[39] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *CVPR*, 2016, pp. 3213–3223.

[40] C. Sakaridis, D. Dai, and L. Van Gool, "Guided curriculum model adaptation and uncertainty-aware evaluation for semantic nighttime image segmentation," in *ICCV*, 2019, pp. 7374–7383.

[41] ——, "ACDC: The adverse conditions dataset with correspondences for semantic driving scene understanding," in *ICCV*, 2021, pp. 10 765–10 775.