The Over-Certainty Phenomenon in Modern UDA Algorithms

Fin Amin
North Carolina State University
samin2@ncsu.edu

Jung-Eun Kim[*]
North Carolina State University
jung-eun.kim@ncsu.edu

Abstract

When neural networks are confronted with unfamiliar data that deviate from their training set, this signifies a domain shift. While these networks output predictions on their inputs, they typically fail to account for their level of familiarity with these novel observations. This challenge becomes even more pronounced in resource-constrained settings, such as embedded systems or edge devices. To address such challenges, we aim to recalibrate a neural network's decision boundaries in relation to its cognizance of the data it observes, introducing an approach we coin as certainty distillation. While prevailing works navigate unsupervised domain adaptation (UDA) with the goal of curtailing model entropy, they unintentionally birth models that grapple with calibration inaccuracies - a dilemma we term the over-certainty phenomenon. In this paper, we probe the drawbacks of this traditional learning model. As a solution to the issue, we propose a UDA algorithm that not only augments accuracy but also assures model calibration, all while maintaining suitability for environments with limited computational resources.

1 Introduction

When encountering new environments, humans naturally adopt a cautious approach, assimilating the novelty to guide their decision-making. This inherent ability to assess unfamiliarity and adjust certainty has not been entirely emulated in artificial neural networks. Unlike humans who might exhibit hesitation in unknown situations, many unsupervised domain adaptation (UDA) algorithms lack an explicit mechanism to modulate certainty in response to the novelty or unfamiliarity of their inputs.

Deep learning has never been a stranger to the challenges of uncertainty. Over the past few years, the miscalibration problem of modern neural networks has gained substantial attention, as highlighted by works such as [1], [2], [3], and [4]. However, there is an observed void in the landscape of unsupervised domain adaptation techniques, with most of them neglecting model calibration during adaptation processes. In this paper, we introduce the *over-certainty phenomenon* which harms model calibration, and propose an algorithm that extends the notion of unfamiliarity - analogous to what humans experience - to UDA.

A prevailing strategy among UDA algorithms is the minimization of entropy, either as an explicit target or as an inherent by-product of their methodology. And while this might bolster accuracy metrics, our research indicates a concerning trend: excessive entropy reduction can be detrimental to model calibration. What makes this trend more problematic is that it occurs within the context of a new domain, where epistemic uncertainty should typically be greater.

To further frame our discussion, UDA is used when a model, trained on a source domain, is presented with the challenges of a different yet analogous target domain. The nuances between these domains,

[*] Correspondence

Preprint. Under review.

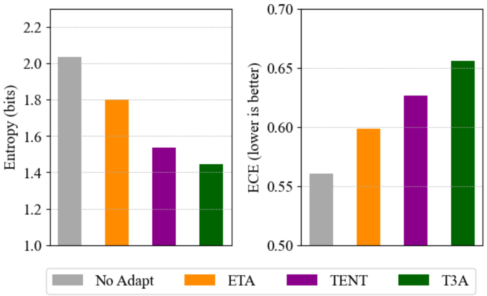


Figure 1: This chart shows adapting a MobileNet to the level 5-intensity *contrast* domain via three modern UDA algorithms. Minimizing entropy is a common objective in recent work. However, this can have consequences on model calibration.

commonly termed as domain shift, can introduce significant disruptions in model performance. UDA, in its essence, aspires to adapt the insights harvested from the source domain and apply them proficiently to the target domain, bypassing the need for labeled data in the latter. Thus, within the context of this domain shift, it can be especially problematic to be too certain.

With the purpose of addressing these intertwined challenges, we introduce *certainty distillation*. This UDA technique seeks to augment accuracy, improve model calibration, and maintain compatibility to resource-constrained devices. By interweaving calibration into the core learning process, we produce a UDA algorithm that jointly improves accuracy and epistemic uncertainty. Furthermore, we introduce a hyperparameter of our algorithm which provides a trade-off between adaptation performance and resource consumption. To summarize, our contributions are:

- The identification of the over-certainty phenomenon in modern UDA algorithms. We provide thorough empirical evidence which corroborates our claims. Furthermore, we provide plausible explanations as to why this happens.
- Certainty Distillation, a new UDA algorithm that achieves SOTA calibration in all of the four datasets and SOTA accuracy uplifts in the majority of domain shifts. Additionally, our algorithm provides favorable memory-consumption vs. performance trade-offs.

2 Related Work and State-Of-The-Arts

We divide our literature survey into three sections. The first section covers methodologies catered towards updating a neural network on unlabeled data. For the sake of brevity, we will refer to unlabeled data as “observations." This section gives an overview of the work done to improve networks on the fly. We start by introducing earlier work, such as dictionary learning techniques, and lead our way into recent developments. The second section covers how we measure neural network calibration and certainty. The third section covers OOD detection.

2.1 Self-Taught Learning and UDA

The phrase “self-taught learning" was coined by [5]. In this work, the authors utilize observations to find an optimal sparse representation of said observations. More precisely, the authors utilize observations to find a set of basis vectors and corresponding activations. By doing this, the authors create a methodology for finding a sparse representation of inputs. Upon finding the basis vectors, the authors then solve for the activations using the training set. Finally, this sparse representation is used to train their model in lieu of the ordinary training set.

Work in this field has extended to a variety of approaches. For example, [6] formulate a similar methodology for utilizing the observations made from edge devices. Specifically, they propose a methodology using SVMs to find so-called “domain invariant features.” These are features that maximize the margin across various domains. Other examples include the use of pseudo-labeling to exploit the existing model’s predictions as target labels [7]. Pseudolabeling can be thought of under the guise of Knowledge distillation (KD) [8, 9]. KD is a transfer learning paradigm where a large neural network, known as the teacher, transfers "knowledge" to a smaller "student" network. Succinctly, the student is trained to match the output of the teacher when given the same input as the teacher [10]. Another technique is ensembling various source-specific networks [11, 12].

More recent advancements include TENT [13], EATA/ETA [14], and T3A [15]. The TENT algorithm focuses on test-time entropy minimization. In other words, the algorithm works by using gradient descent to minimize:

$$L_{TENT} = -\sum_{y \in \mathcal{C}} f_\Theta(y|x) \log f_\Theta(y|x) \tag{1}$$

to update the model's batch-normalization parameters. ETA[2] advances on TENT by making sure that observations are *reliable* and *non-redundant* before they are used for updating the batch-normalization parameters. To do this, they compute a sample adaptive weight, $\mathcal{S}(x)$, for each observation before minimizing entropy:

$$L_{ETA} = -\mathcal{S}(x) \sum_{y \in \mathcal{C}} f_\Theta(y|x) \log f_\Theta(y|x) \tag{2}$$

Where $\mathcal{S}(x)$ is a function of the entropy of the model towards the batch sample (i.e., the reliability) and the similarity to what it has seen before (i.e., non-redundancy).

The T3A algorithm [15] differs from the previous two as it focuses on updating the *prototypes* [16] of each class during test time:

$$S_k^t = \begin{cases} S_k^{t-1} \cup \{f_\theta(x)\}, & \text{if } \hat{y} = y_k \\ S_k^{t-1}, & \text{else.} \end{cases} \tag{3}$$

$$c_k = \frac{1}{|S_k|} \sum_{z \in S_k} z \tag{4}$$

where c_k represents the centroid of the prototypes of a class k. To inference, T3A computes:

$$\text{argmax}_{y_k} \gamma_c(Y = y_k | f_\theta(x)) = \frac{\exp(z \cdot c_k)}{\sum_j \exp(z \cdot c_j)} \tag{5}$$

where z is the output of the feature extractor.[3] Unlike TENT or ETA, T3A does not use back propagation. However, similar to ETA, this algorithm filters less reliable samples during equation 3 by only keeping the M most reliable (low-entropy) prototypes for each class. Therefore, the algorithm stores $C \cdot M$ prototypes, where C is the number of classes.

2.2 Neural Network Calibration

Neural network calibration has been of intense interest in recent years. The accuracy of the confidence of a neural network is extremely important—as confidence values, reflecting the probability assigned to a prediction, are used in a variety of domains. For example, the authors of BranchyNet [17] utilize neural network confidence values to allow an early exit for faster inference, banking on high confidence at intermediate layers. Conversely, [18] remarks on the lack of certainty calibration in most deep networks, where certainty encompasses not just confidence but also the model's overall assurance in its predictions. In our work, we measure certainty as $H^{-1} = (\texttt{Entropy}_2)^{-1}$. The most remarkable observation they found was that most models are either *too confident* or *not confident enough*, possibly due to overfitting during training.

The authors of [19] explore this concept further. They measure a model's expected calibration error (ECE) with respect to changes (rotation, translation, etc) in the test set. ECE is defined as:

[2] While the authors of EATA/ETA introduce two similar algorithms, for our paper, we focus on ETA.

[3] We define the feature extractor as all the layers of the backbone before the final dense layer. The final dense layer, which has a size that is a function of the number of classes, is what we refer to as the classifier.

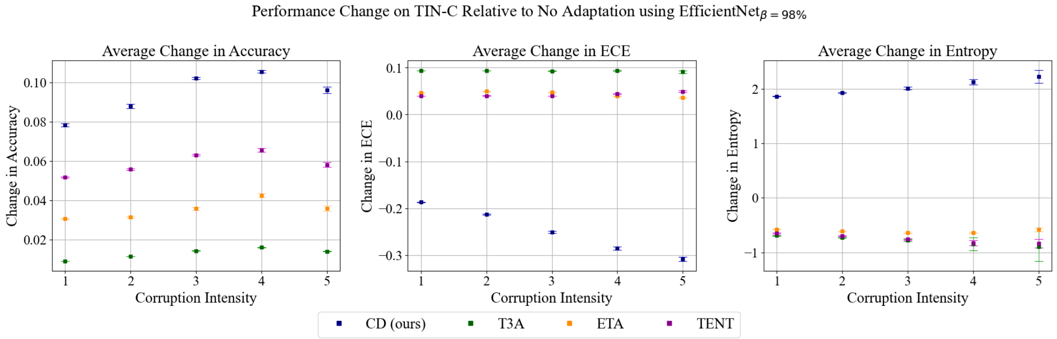


Figure 2: A potential cause of the over-certainy phenomenon is that prior work aims at increasing certainty towards observations despite the domain shift (corruption intensity) increasing. Our algorithm, certainty distillation, achieves state-of-the-art accuracy and calibration on TinyImageNet-C. The results are averaged over 15 unique domain shifts; variances across the domains are shown by the error bars.

$$\text{ECE} = \sum_{i=1}^{N} \frac{|B_i|}{N} |\text{accuracy}(B_i) - \text{confidence}(B_i)| \tag{6}$$

They notice models calibrated on the validation set tend to be well calibrated on the test set, but are not properly calibrated to shifted data. Recent work has investigated this phenomenon. [1] approaches the issue through a method known as *temperature scaling* while [20] approaches this problem by regularizing the logit norm. More classical solutions to this problem exist as well; [21] and [22] consider label smoothing to address this issue.

2.3 Detecting Out of Distribution Data

There has been intense interest in recent years in the problem of determining if, and to what degree an observation is similar to what a model was trained on. The authors of [23] observe that if an autoencoder were trained to reconstruct inliers, it would have a greater reconstruction error when reconstructing OOD data. [24] and [25] approach this issue by observing that the discriminator of a GAN learns whether or not a given input is an inlier. Many other works delve into this domain; OpenMax [26] analyzes mean activation vectors (MAV), and [26] investigates the optimal recognition error. Other examples include [27, 28, 29, 30, 31]. Literature such as [32] aims at determining under what conditions OOD detection is possible.

3 Proposed Approach

3.1 The Over-Certainty Phenomenon (OCP)

In this work, we present evidence for what we dub the *over-certainty phenomenon* (OCP) of contemporary UDA approaches. This phenomenon is that UDA algorithms tend to miscalibrate their underlying backbone networks by causing their predictions to be excessively certain. Modern UDA algorithms often strive to decrease test-time entropy. However, as shown in Fig. 1, this entropy reduction may increase ECE because the models become overly certain on their predictions.

This phenomenon of existing algorithms causing models to become overly certain presents itself across many other datasets. For example, in Table 1a, T3A reduces entropy by a factor of about 4 in the *Art, Clipart* and *Product* domains. As before, it causes ECE to worsen compared to the baseline. In addition to the results we show in this paper, we provide extensive evidence on this phenomenon in the Appendix. We do not claim that UDA algorithms should *always* strive to increase backbone uncertainty; poor calibration can also be caused by under-certainty. In fact, there exist cases where reducing entropy compared to baseline improves calibration. However, we find that the resulting calibration is still sub-optimal. Despite these complexities, our investigation reveals a consistent pattern: *the over-certainty phenomenon causes sub-optimal model calibration*, a significant concern for safety, robustness, and reliability.

3.2 Towards Better Calibration in UDA: Discussion on SOTA Algorithms

We identify two plausible causes of the OCP, the first issue is that modern UDA algorithms aim at minimizing backbone entropy too aggressively. In the case of TENT and ETA, their loss functions, equations (1) and (2), explicitly aim at reducing a model's entropy. Regarding TENT, there is no regularization of this process. In the case of ETA, the algorithm uses a *reliability score*, $S(x)$, which aims at weighing observations differently but does not regularize the distributions of the pseudo-labels. T3A does not explicitly reduce entropy as it does not use a loss function, but the authors claim this as an effect of using their algorithm. In fact, they show in certain datasets T3A reduces entropy more than TENT does.

Another issue is how existing methods evaluate observation *reliability*, the suitability of a model's prediction for use for adaptation. Previous works, ETA and T3A, tap into the power of model certainty, using it to weigh the influence of observations. ETA assesses reliability by ensuring that observations meet a certain entropy threshold; similarly, T3A uses entropy to sort the importance of class prototypes. However, there are drawbacks in using entropy as a proxy for reliability in this manner [20]. To illustrate our point, we give a toy example of how using entropy can lead to a misleading conclusion:

Example 1 *Suppose that we analyze the classifier while classifying between two classes with class centroids, c_0 and c_1. This is done by taking the output of the feature extractor, $f(x) = z$, and computing the dot product between the centroids and z.*

$$g = [z \cdot c_0, z \cdot c_1] \tag{7}$$

Consider g_s and g_{t1}, g_{t2} as vectors representing the inner products related to a specific training sample and observation, respectively. Specifically, g_s corresponds to the inner products with the training sample where $f(x_s) = z_s$, and g_{t1}, g_{t2} correspond to the inner products with the observation where $f(x_t) = z_t$. For the sake of an example, let's assume:

$$g_s = [8.0, 7.29]; \quad g_{t1} = [.9199, .00019]; \quad g_{t2} = [6.1, 6.5];$$

If we take the softmax of these vectors and compute the entropy, we get $\texttt{Entropy}_2(SM(g))$ *for g_s, g_{t1} and g_{t2}, as 0.92 bits, 0.86 bits and 0.97 bits, respectively.*

Notice that if we consider the entropy of these three vectors as a proxy for reliability, we would consider x_{t1} to be more reliable than x_{t2}, despite x_{t2} having considerably greater average inner product with the class centroids. It is far more likely that the values of g_{t1} occurred due to spurious feature correlations between x_{t1} and the class centroids. In fact, in the scenario above, x_{t1} would be deemed to be more reliable than the genuine source domain observation x_s. Furthermore, there is no consideration of the source domain's certainty. By only evaluating the target domain's certainty without juxtaposing it against the source domain certainty, there is a lack of reference in terms of assessing the reliability of the observation.

3.3 The Certainty Distillation Algorithm

To ameliorate the over-certainty phenomenon, we introduce certainty distillation (`CD`) (Algorithm 2). The `CD` algorithm employs a novel adaptation technique to strategically manipulate model certainty to the unknown to improve calibration. `CD` refines the model's certainty levels, aligning them more closely with its actual accuracy, by selectively adjusting the temperature parameter during the distillation process. This is achieved without directly altering ground truth labels, instead focusing on the tempering of logits through temperature adjustments. The algorithm employs a two-model approach, using a teacher model to guide the calibration of a student model, with an emphasis on preventing over-certainty and achieving better model calibration.

This process involves iterative adjustments of the student model's predictions, guided by the comparative analysis of entropy and logit norms, thereby fostering a more accurate and reliable predictive model. The inputs N_{te}, N_s, H_0, and X correspond with the teacher model, the student model, the student's average entropy on the training set, and unlabeled observations, respectively. An important detail of our methodology is the student and teacher are two copies of the same model. The H_0 parameter is the backbone's average entropy on the training set. It is used as a reference point; the idea is to compare the model's certainty on X with respect to the certainty of what it was trained on. The input κ is the median l_2 norm of the training-set logits; this gives us a context in terms of logit norms. The `Compute_T` (Algorithm 1) returns a temperature for each observation, T_{vec}, with respect to observation entropy and relative logit-norm. Parameters T_{min}, T_{max}, and H_{max} are used to scale the resulting temperatures; for our experiments we use 1.2, 5.0, and $\log_2(C)$ respectively unless stated otherwise. Lastly, the λ parameter is the learning rate for SGD, which we set to 0.001 for all experiments.

Algorithm 1 `Compute_`τ

```
Input: $N_{te}(X), H_0, H_{max} T_{min}, T_{max}, \kappa$
Output: $T_{vec}$
$z_{logits} = N_{te}(X)$
$H_{vec} = \texttt{Entropy}_2(z_{logits})$ {entropy for each sample}
$scaled\_H_{vec} = \texttt{sigmoid}(H_{vec} - H_0)$
Init. $T_{vec}$
for $e_i \in scaled\_H_{vec}$ do
    $t_i = T_{min} + \left(\frac{e_i}{H_{max}}\right) \cdot (T_{max} - T_{min})$
    $\tau_i = [1 + \frac{2}{5}(1.5 - 1.5 \cdot \texttt{sigmoid}(\frac{|z_{logits}|_2}{\kappa}))] \cdot t_i$
    Store $T_{vec} \leftarrow \tau_i$
end for
return $T_{vec}$
```

To optimize memory efficiency, we freeze a large subset, β, of the weights of our student so that we only have to store the weights of the teacher network plus the weights which we choose not to freeze. The `Freeze_b_Layers`$(\mathtt{N_s}, \beta)$ function works by freezing the parameters of N_s. For notational purposes, we define β as the percentage of the backbone model that is frozen. We found empirically that freezing last (i.e., the layers closest to the output) b layers works the best. We select b so that β percent of our backbone model is frozen. Therefore the b value will vary across backbones for a target β value. In other words, to select b, one should compute the number of parameters in each layer and select b layers such that target percentage of network parameters, β is frozen.

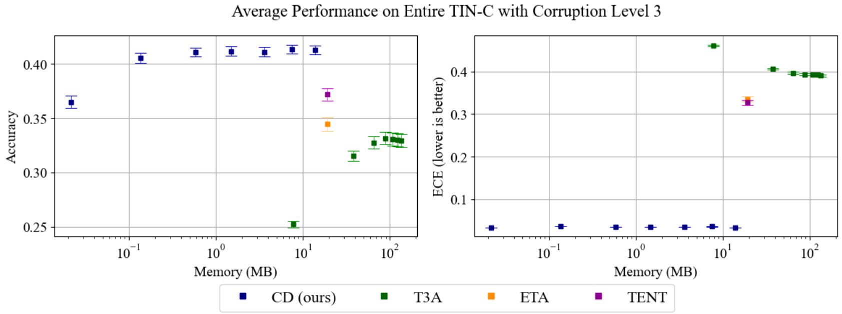


Figure 3: Memory consumption trade-off. Results were produced by varying the β and M parameters of CD and T3A and allowing the models to adapt using EfficientNet. We found domain-to-domain $\sigma_{\mathtt{max}} = \mathtt{52.07MB}$ for the memory consumption of T3A with respect to M.

An interesting interpretation as to how our model improves accuracy is through the works of [33] and [34]. Although the former's work concerns itself in the semi-supervised learning setting, we found their observations to be relevant. That is, they introduce the *noisy student,* a network that has been *noised* by dropout and stochastic-depth. They find that their noisy student can even learn to outperform the teacher which initially produced the pseudo-labels. For the latter work, they establish that label smoothing mitigates label noise, which is a desirable property with respect to unsupervised adaptation. Specifically, they find that label smoothing can be thought of as a regularizer. This motivates us to smooth more aggressively when we notice that an observation might be less reliable.

Algorithm 2 `Certainty_Distillation`

```
Input: $N_{te}, N_s, H_0, X, \kappa$
Parameter: $T_{min}, T_{max}, H_{max}, \beta, \lambda$
Output: $N_s^+$
$N_s = \texttt{Freeze\_b\_Layers}(\mathtt{N_s}, \beta)$
$T_{vec} = \texttt{Compute\_}\tau(\mathtt{N_{te}}(\mathtt{X}), \mathtt{H_0}, \kappa, \mathtt{H_{max}}, \mathtt{T_{min}}, \mathtt{T_{max}})$
$T_{avg} = \texttt{avg}(T_{vec})$
Init $loss$
for $x_i \in X$ and $\tau_i \in T_{vec}$ do
    $s_{si} = \texttt{SoftMax}_{\mathtt{T=1}}(\mathtt{N_s}(\mathtt{x_i}))$
    $s_{ti} = \texttt{SoftMax}_{\mathtt{T=\tau_i}}(\mathtt{N_t}(\mathtt{x_i}))$
    $l_{CD} = T_{avg}^2 \cdot BCE(s_{si}, s_{ti})$
    Store $loss \leftarrow l_{CD}$
end for
$L_{CD} = \texttt{avg}(loss)$
$N_s' \leftarrow \theta_s - \lambda \nabla L_{CD}(\theta_s)$
$N_s^+ = \texttt{Temperature\_Scale}(\mathtt{N_s'}, \mathtt{T_{avg}})$
return $N_s^+$
```

Algorithm	Art	Clipart	Product	Real World
No Adaptation	0.9902	0.9992	0.5663	0.0113
CD (ours)	2.2642	2.3637	1.5137	0.0892
T3A	0.2527	0.2638	0.1424	0.0837
ETA	0.7716	0.6305	0.4086	0.0118
TENT	0.7820	0.6328	0.4164	0.0077

(a) Shannon Entropy on Home Office.

Algorithm	Art	Clipart	Product	Real World
No Adaptation	0.3133	0.2818	0.1940	0.0024
CD (ours)	**0.1107**	**0.0715**	**0.0599**	0.0072
T3A	0.4330	0.3666	0.2245	0.0427
ETA	0.3341	0.3219	0.2136	**0.0023**
TENT	0.3281	0.3049	0.2065	0.0025

(b) ECE on Home Office (lower is better).

Table 1: Our investigation reveals a pattern of existing UDA algorithms achieving sub-optimal calibration. We suspect this is caused by excessive entropy minimization. Experiment done using the EfficientNet$_{\beta=0\%}$ backbone on the Home Office dataset. Note that ECE values less than 0.01 are considered already well calibrated [1].

The τ parameter returned by Algorithm 1 plays a pivotal role, we name it the *certainty regularizer*. It regulates the "sharpness" of predicted probabilities and smoothens the predictions produced by the teacher. By preventing the model from becoming inappropriately certain in its predictions, we produce a model that is better calibrated — its prediction certainty more closely aligns with its true accuracy. In CD, we do not label smooth directly, but instead adjust the temperature parameter of our teacher during the distillation process. To show how CD regularizes observations appropriately, we continue from example 1:

Example 2 *Given the same g_{t1}, g_{t2} and g_s from example 1, we input these into our* `Compute_`τ *algorithm. We set $H_0 = H(g_s), \kappa = |g_s|_2, T_{min} = 1.2$ and $T_{max} = 4.0$. Our algorithm first computes a scaled entropy, $scaled_H g$ with respect to the source domain entropy for g_s, g_{t1} and g_{t2}, as 0.50, 0.49, and 0.27, respectively.*

In step 6 of `Compute_`τ*, this entropy is transformed into a preliminary regularizer, t_i. Afterwards, step 7 adjusts t_i by considering logit norm with respect to the source-domain logit norm.*

$$\tau_{g_s} = 2.39; \quad \tau_{g_{t2}} = 2.43; \quad \tau_{g_{t1}} = 2.57$$

Notice that, unlike purely entropy-based methods, the `Compute_`τ algorithm correctly assigns greater regularization to the less reliable samples. Namely, step 7 ensures that samples that are low-entropy due to degenerate reasons are properly regularized by considering logit norm. Furthermore, unique from existing algorithms, our regularizer directly addresses model certainty. The impact of `Compute_`τ is analyzed in Fig. 4.

4 Experiments

4.1 Experimental Setting

In order to evaluate CD, we conduct a series of experiments using three different backbone models across five datasets. Our primary evaluation metrics will be model accuracy and ECE$_{bins=15}$ on the observations, allowing us to examine both the predictive performance and the calibration quality of the models. By using varied domains and different backbone architectures, we aim to demonstrate the robustness and adaptability of our algorithm in handling diverse and challenging unsupervised domain adaptation scenarios. All experiments are run four times using $\texttt{random\_seed} = 0, 1, 2, 3$, respectively. Run-to-run variances are very low; the ones which we do not show in the main paper are reported in the appendix. Note that the β parameter presented is only relevant to CD; all other algorithms used the vanilla version of the respective backbones.

4.2 Datasets

The following publicly available UDA datasets are used in our experiments; we selected these because they are commonly used in existing works and provide a variety of domain shifts. In total, we evaluate our algorithm over 26 domain shifts. Furthermore, 15 of our domain shifts have 5 levels of corruption attributed to them. Dataset preprocessing steps are written in more detail in the appendix. For some datasets, we tested using the "leave one out" (LOO) paradigm; for example, in PACS, to test generalization to *pictures*, we first trained our backbone networks on *art, cartoon, sketch* before adapting. For TIN-C, we first trained on a source/corruption-free domain and adapted to some domain shift.

1. PACS [35] has 4 domains: *pictures, art, cartoon, sketch* with 7 classes. Tested using LOO.
2. HomeOffice [36] has 4 domains: *art, clipart, product, real* with 65 classes. Tested using LOO.
3. Digits is a combination of 3 "numbers" datasets: USPS [37], MNIST [38], and SVHN [39]. There are 10 classes. Tested using LOO by training on the source domains' training sets and adapting to target domain's test set. For this experiment, we set T_{min} and T_{max} parameters to 1.05 and 4.0 respectively.
4. TinyImageNet-C (TIN-C) [40], has 15 domains with 200 classes. Backbones are trained on corruption-free (source) training set, adapted to and evaluated on corrupted (target) domains. For each target domain, there are 5 tiers of corruption.

4.3 Back Bones and Training Details

We test all but the Digits dataset on two popular low-resource classifiers, EfficientNetB0 [41] and MobileNet [42] pre-trained for ImageNet [43]. We flatten the output of both networks and add a final dense layer with an output shape equivalent to the number of classes.

We evaluate the Digits dataset using SmallCNN, a custom lightweight network tailored to handle grayscale images, serving as a representative of more compact and straightforward architectures for less complex datasets. SmallCNN encompasses a variety of essential building blocks, including 2D convolutional layers equipped with different filters, batch normalization, ReLU activation, and max-pooling layers. The network also integrates dense layers and a final classifier layer to make predictions for the given number of classes. The specific details and orderings of the layers in SmallCNN are elaborated on in the appendix. Note that all three models use batch normalization layers.

4.4 State-Of-The-Art Approaches for Comparison

We compare against TENT, T3A and ETA, the three most recent UDA algorithms, and also a baseline with no adaptation, (No Adapt). For ETA, we set $\texttt{E\_0} = 0.4 \cdot \ln(\texttt{C})$, as this was their recommended value, and $\epsilon = \{0.6, 0.1, 0.4, 0.125\}$ for each enumerated dataset, respectively. These ϵ values were empirically chosen to help their performance. For T3A, we set the number of supports to retain, $M = \infty$, as this provides the lowest calibration error. We do a single iteration of adaptation for

Table 2: Average accuracy, ECE, and entropy on the Digits dataset using the SmallCNN$_{\beta=0\%}$ backbone. Domain-to-domain $\sigma^2_{\mathtt{max}} = [0.22, 0.18, 0.20]$ for accuracy, ECE and entropy, respectively

Algorithm	Accuracy	ECE	Entropy
No Adaptation	0.5894	0.3014	0.4386
CD (ours)	**0.6475**	**0.1685**	1.2195
T3A	0.6247	0.2713	1.8729
ETA	0.6463	0.2698	0.3561
TENT	0.6445	0.2617	0.3980

all algorithms unless stated otherwise. We use a $\mathtt{batchsize} = 50$ for CD and use the authors' recommended batch sizes for the rest.

4.5 Results

We present our accuracy and ECE measurements on the four aforementioned datasets. To show evidence of the over-certainty phenomenon, we also report prediction entropy. More comprehensive figures can be found in the appendix. We also investigate the impact of parameter β, the amount of the student frozen for the CD algorithm, and compare it to the impact of parameter M of the T3A algorithm in Fig. 3. Doing this allows us to investigate the trade-offs between domain adaptability and resource consumption. Furthermore, we perform an ablation study on our algorithm. We measure the impact of our $\mathtt{Compute\_}\tau$ function, which produces our certainty regularizer τ.

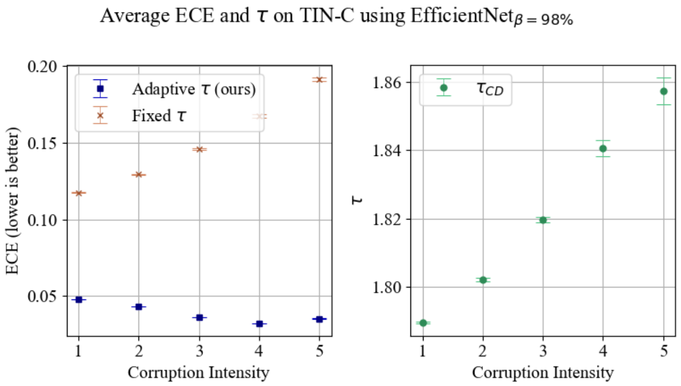


Figure 4: Our ablation study highlights the effectiveness of the $\mathtt{Compute}_\tau$ algorithm compared to a fixed τ optimized for minimal ECE on the source-domain training set. The rightmost figure displays the τ values generated by $\mathtt{Compute\_}\tau$, with error bars indicating domain-to-domain variance.

4.6 CD Reduces Calibration Error

Due to our algorithm addressing the over-certainty phenomenon, we significantly improve calibration performance; CD had the lowest average ECE in all tested datasets and in nearly all individual domain shifts. We recognize that reducing entropy *did* improve calibration compared to baseline in some cases in Tables 2 and 3, but the resulting calibration was still sub-optimal. Fig 4 empirically validates our finding that an adaptive certainty regularizer aids in reducing ECE.

4.7 CD Augments Accuracy

In addition to strong calibration performance, CD provides consistent accuracy uplifts while not necessitating any transformations on observations. According to Fig. 2 our algorithm gives significant accuracy improvements on a variety of domain shifts without requiring any alterations to initial

Table 3: Average accuracy, ECE, and entropy on the PACS dataset using the MobiletNet$_{\beta=0\%}$ backbone. Domain-to-domain $\sigma^2_{\texttt{max}} = [0.01, 0.02, 0.12]$ for accuracy, ECE and entropy respectively.

Algorithm	Accuracy	ECE	Entropy
No Adaptation	0.8410	0.1110	0.1768
CD (ours)	0.8482	**0.0754**	0.3660
T3A	**0.8567**	0.1160	0.0960
ETA	0.8541	0.1011	0.1692
TENT	0.8483	0.1079	0.1618

source-domain training. Across nearly all domain shifts, backbone models, and corruption intensities within TIN-C, CD has the highest accuracy and lowest ECE. Furthermore, it performs competitively against the existing state of the art in the remaining datasets while addressing calibration.

4.8 CD is Suitable for Low Resource Scenarios

Our experiments with SmallCNN in Table 2 show our algorithm's relevance for low-resource scenarios. We achieve strong accuracy while maintaining low calibration error. In our experiments using EfficientNet, we were able to freeze the majority of the parameters of our backbone while still achieving significant performance improvements in both accuracy and ECE. Furthermore, our methodology does not require computing distance metrics between our source domain and target domain - which would increase computational complexity. The model weights which are trainable are what predominantly consume memory for our algorithm. Therefore, the memory overhead of CD, P_{CD}, is controlled by β and is at most the trainable parameters of the backbone model. The memory overhead of T3A, P_{T3A}, is a function of their hyperparameter M and the size of the final classifier. More formally, if we define f_ω, c_ω as the number of parameters in the feature extractor and classifier respectively:

$$\mathrm{P}_{T3A} = M \cdot c_\omega \tag{8}$$

$$\mathrm{P}_{CD} = (1 - \beta) \cdot (f_\omega + c_\omega) \tag{9}$$

$$\mathrm{P}_{ETA} = \mathrm{P}_{TENT} = \texttt{NumBNParams} \tag{10}$$

In Fig. 3 we show that our method is memory efficient while maintaining competitive calibration and accuracy. In fact, we can achieve significant performance uplifts using $\leq$ `1MB` of extra memory.

5 Discussion

In this work, we identify the *over-certainty phenomenon* of state-of-the-art UDA methodologies which cause harm to model calibration. To ameliorate this issue, we introduce a certainty regularizer, τ, which adapts the sharpness of self-labels and persuades overall model entropy. The resulting algorithm, CD, jointly improves model accuracy and reduces calibration error while remaining memory efficient. Another merit of our methodology is its compatibility with the majority of backbone networks. CD does not require batch normalization layers like TENT and ETA do. This model agnostic approach permits greater freedom when choosing a suitable backbone. Furthermore, CD is compatible with existing prototypical learning approaches such as T3A and the work done by [16]. In the interest of reproducibility, we provide the code for our algorithm in the supplementary materials.

References

[1] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[2] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U. Rajendra Acharya, Vladimir

Makarenkov, and Saeid Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

[3] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[4] Anusri Pampari and Stefano Ermon. Unsupervised calibration under covariate shift. *arXiv preprint arXiv:2006.16405*, 2020.

[5] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766, 2007.

[6] Fei Ma, Chengliang Wang, and Zhuo Zeng. Svm-based subspace optimization domain transfer method for unsupervised cross-domain time series classification. *Knowledge and Information Systems*, pages 1–29, 11 2022.

[7] Dong-Hyun Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896. Atlanta, 2013.

[8] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[9] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[10] Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A Alemi, and Andrew G Wilson. Does knowledge distillation really work? *Advances in Neural Information Processing Systems*, 34:6906–6919, 2021.

[11] Massimiliano Mancini, Samuel Rota Bulo, Barbara Caputo, and Elisa Ricci. Best sources forward: domain generalization through source-specific nets. In *2018 25th IEEE international conference on image processing (ICIP)*, pages 1353–1357. IEEE, 2018.

[12] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip Yu. Generalizing to unseen domains: A survey on domain generalization. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[13] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020.

[14] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International conference on machine learning*, pages 16888–16905. PMLR, 2022.

[15] Yusuke Iwasawa and Yutaka Matsuo. Test-time classifier adjustment module for model-agnostic domain generalization. *Advances in Neural Information Processing Systems*, 34:2427–2440, 2021.

[16] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[17] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *CoRR*, abs/1709.01686, 2017.

[18] Weicheng Zhu, Matan Leibovich, Sheng Liu, Sreyas Mohan, Aakash Kaku, Boyang Yu, Laure Zanna, Narges Razavian, and Carlos Fernandez-Granda. Deep probability estimation, 2022.

[19] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift, 2019.

[20] Hongxin Wei, Renchunzi Xie, Hao Cheng, Lei Feng, Bo An, and Yixuan Li. Mitigating neural network overconfidence with logit normalization. In *International Conference on Machine Learning*, pages 23631–23644. PMLR, 2022.

[21] Chang-Bin Zhang, Peng-Tao Jiang, Qibin Hou, Yunchao Wei, Qi Han, Zhen Li, and Ming-Ming Cheng. Delving deep into label smoothing. *IEEE Transactions on Image Processing*, 30:5984–5996, 2021.

[22] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? *CoRR*, abs/1906.02629, 2019.

[23] Kai Tian, Shuigeng Zhou, Jianping Fan, and Jihong Guan. Learning competitive and discriminative reconstructions for anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5167–5174, 2019.

[24] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *CoRR*, abs/1703.05921, 2017.

[25] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection, 2018.

[26] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.

[27] Yifei Ming, Ziyang Cai, Jiuxiang Gu, Yiyou Sun, Wei Li, and Yixuan Li. Delving into out-of-distribution detection with vision-language representations. *Advances in Neural Information Processing Systems*, 35:35087–35102, 2022.

[28] Jaewoo Park, Jacky Chen Long Chai, Jaeho Yoon, and Andrew Beng Jin Teoh. Understanding the feature norm for out-of-distribution detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1557–1567, 2023.

[29] Xue Jiang, Feng Liu, Zhen Fang, Hong Chen, Tongliang Liu, Feng Zheng, and Bo Han. Detecting out-of-distribution data through in-distribution class prior. In *International Conference on Machine Learning*, pages 15067–15088. PMLR, 2023.

[30] Xinheng Wu, Jie Lu, Zhen Fang, and Guangquan Zhang. Meta ood learning for continuously adaptive ood detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19353–19364, 2023.

[31] Wenjun Miao, Guansong Pang, Tianqi Li, Xiao Bai, and Jin Zheng. Out-of-distribution detection in long-tailed recognition with calibrated outlier class learning. *arXiv preprint arXiv:2312.10686*, 2023.

[32] Zhen Fang, Yixuan Li, Jie Lu, Jiahua Dong, Bo Han, and Feng Liu. Is out-of-distribution detection learnable? *Advances in Neural Information Processing Systems*, 35:37199–37213, 2022.

[33] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10687–10698, 2020.

[34] Michal Lukasik, Srinadh Bhojanapalli, Aditya Menon, and Sanjiv Kumar. Does label smoothing mitigate label noise? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6448–6458. PMLR, 13–18 Jul 2020.

[35] 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.

[36] 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.

[37] J.J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[38] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[39] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *Neurips Workshop on Deep Learning*, 2011.

[40] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[41] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[42] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[43] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

[45] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[46] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

A Appendix

A.1 Experimental Setup Details

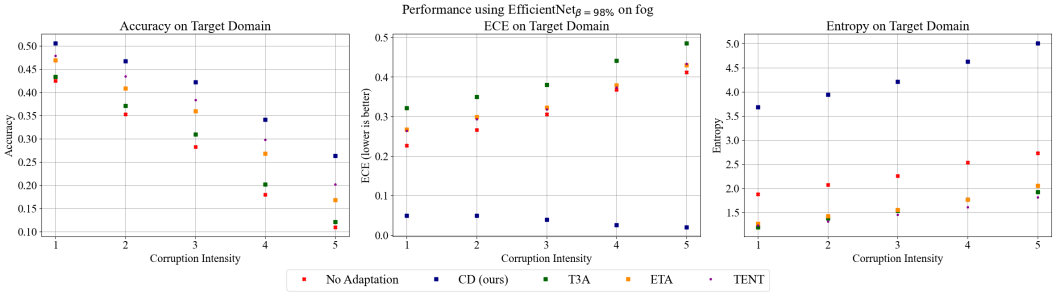
We use TensorFlow 2.9 [44] with Nvidia CUDNN version 11.3 on an RTX 3080 16GB laptop GPU with 32GB of system memory.

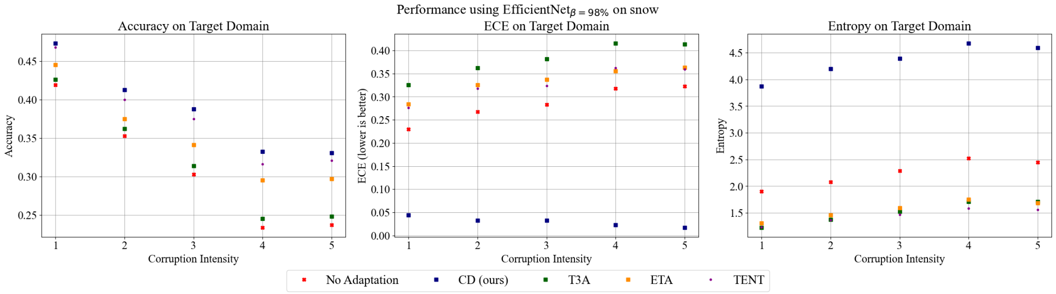
1. PACS [35] has 4 domains: *pictures, art, cartoon, sketch* with 7 classes. All images are resized to $(227, 227, 3)$ and scaled between $[0, 255]$. Tested using LOO.
2. HomeOffice [36] has 4 domains: *art, clipart, product, real* with 65 classes. All images are resized to $(128, 128, 3)$ and scaled between $[0, 255]$. Tested using LOO.
3. Digits is a combination of 3 "numbers" datasets: USPS [37], MNIST [38], and SVHN [39]. The images are resized to $(32, 32, 1)$ and scaled between $[0, 255]$. There are 10 classes. Tested using LOO by training on the source domains' training sets and adapting to target domain's test set. For this experiment, we set T_{min} and T_{max} parameters to 1.05 and 4.0 respectively.
4. TinyImageNet-C (TIN-C) [40], has 15 domains with 200 classes. All images are resized to $(256, 256, 3)$ and scaled between $[0, 255]$. Backbones are trained on corruption-free (source) training set, adapted to and evaluated on corrupted (target) domains. For each target domain, there are 5 tiers of corruption.

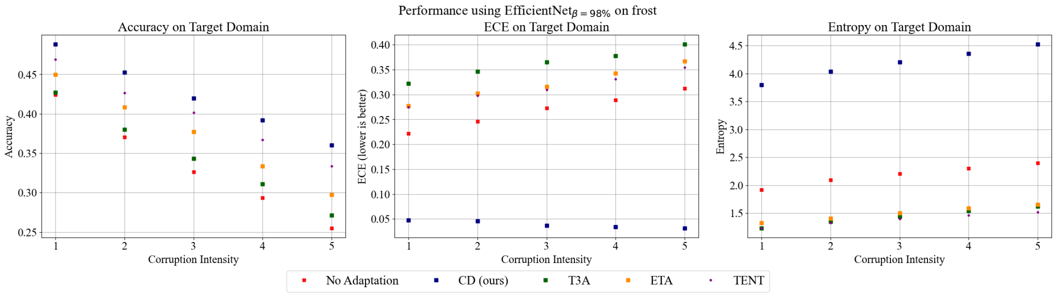
We do most initial training on the source domain using $\texttt{RMS\_Prop}(\texttt{lr} = \texttt{2e} - 4)$ [45] to minimize cross-entropy loss for $\texttt{epochs} = \{15, 15, 5, 25\}$ for each enumerated dataset, respectively. Small-CNN is compiled and initially trained with the Adam optimizer [46] in lieu of RMSProp.

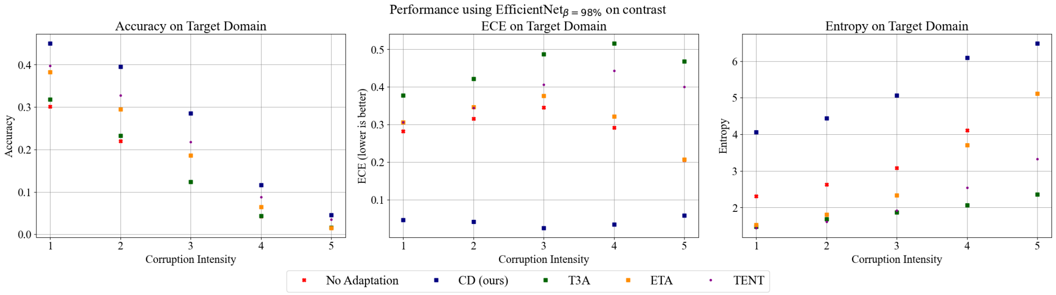
Note that MobileNet expects inputs to be prepossessed in a unique manner. We use Tensorflow's off-the-shelf pre-processing layer for MobileNet at the input.

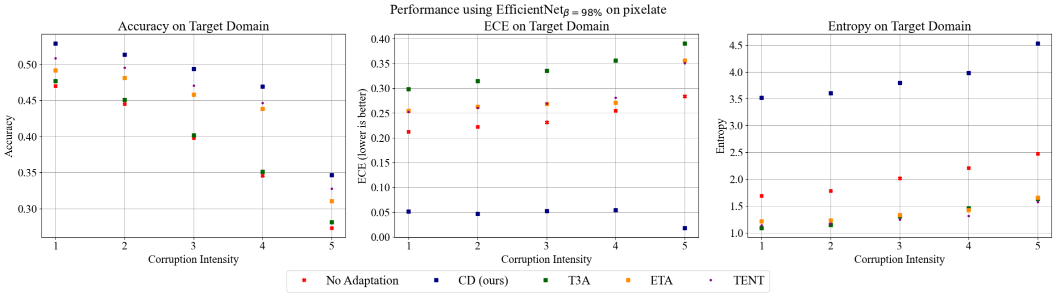
A.2 Results using EffecientNet$_{\beta=98\%}$ on TinyImageNet-C

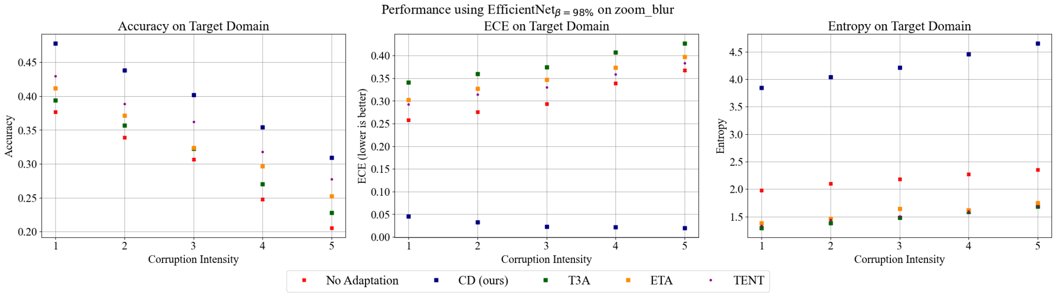


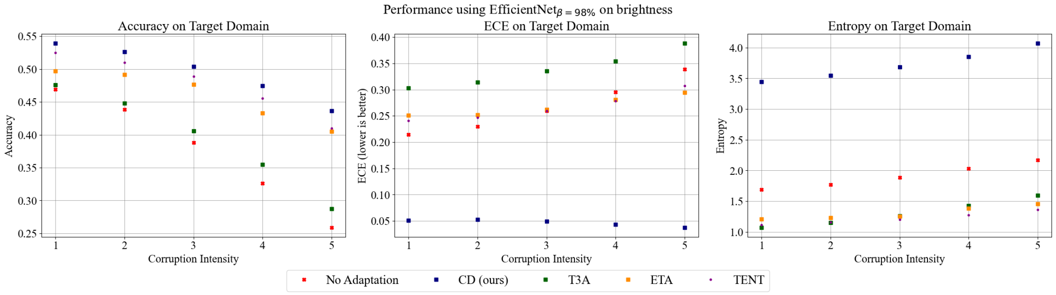


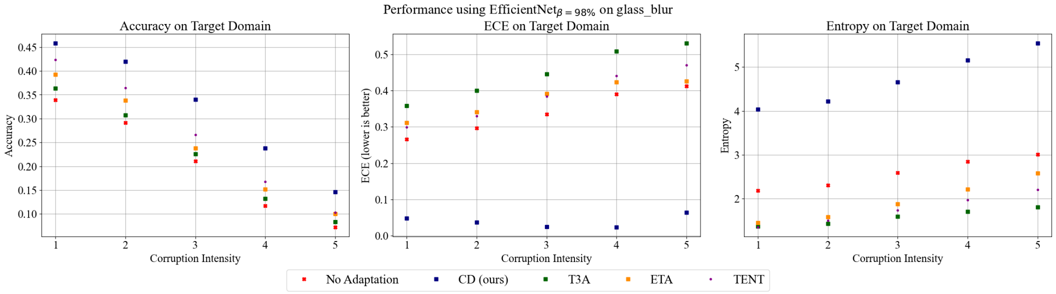

Performance using EfficientNet$_{\beta = 98\%}$ on frost
Accuracy on Target Domain
Accuracy
0.25
0.30
0.35
0.40
0.45
ECE on Target Domain
ECE (lower is better)
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Entropy on Target Domain
Entropy
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Corruption Intensity
1
2
3
4
5
No Adaptation
CD (ours)
T3A
ETA
TENT

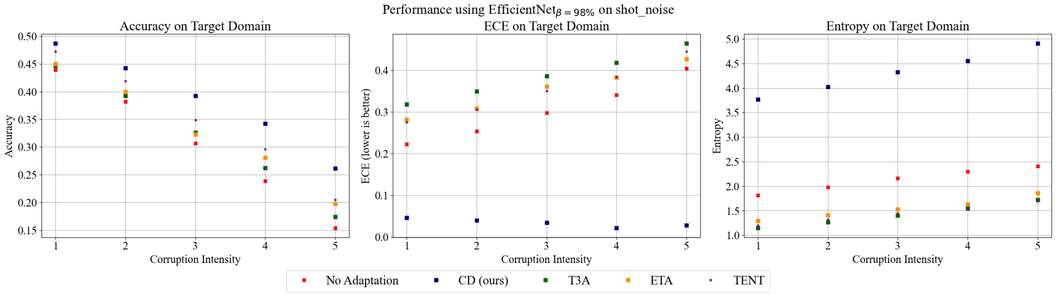

Performance using EfficientNet$_{\beta = 98\%}$ on contrast
Accuracy on Target Domain
Accuracy
0.0
0.1
0.2
0.3
0.4
ECE on Target Domain
ECE (lower is better)
0.1
0.2
0.3
0.4
0.5
Entropy on Target Domain
Entropy
2
3
4
5
6
Corruption Intensity
1
2
3
4
5
No Adaptation
CD (ours)
T3A
ETA
TENT

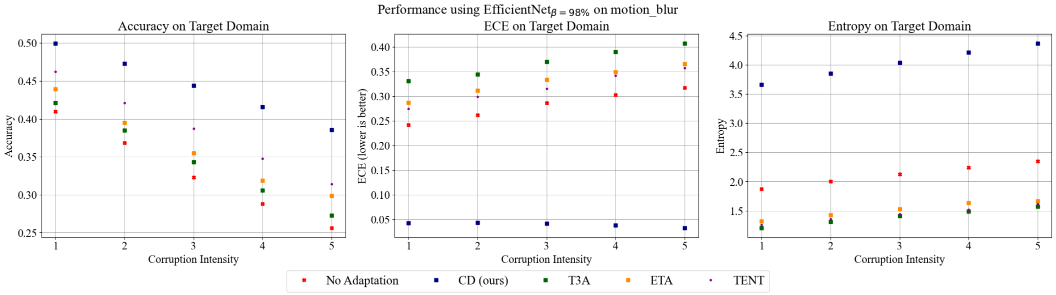

Performance using EfficientNet$_{\beta = 98\%}$ on pixelate
Accuracy on Target Domain
Accuracy
0.30
0.35
0.40
0.45
0.50
ECE on Target Domain
ECE (lower is better)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Entropy on Target Domain
Entropy
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Corruption Intensity
1
2
3
4
5
No Adaptation
CD (ours)
T3A
ETA
TENT

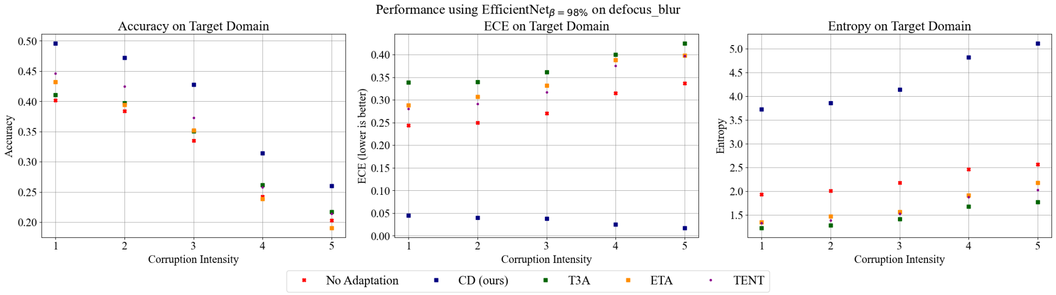

Performance using EfficientNet$_{\beta = 98\%}$ on zoom_blur
Accuracy on Target Domain
Accuracy
0.20
0.25
0.30
0.35
0.40
0.45
ECE on Target Domain
ECE (lower is better)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Entropy on Target Domain
Entropy
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Corruption Intensity
1
2
3
4
5
No Adaptation
CD (ours)
T3A
ETA
TENT

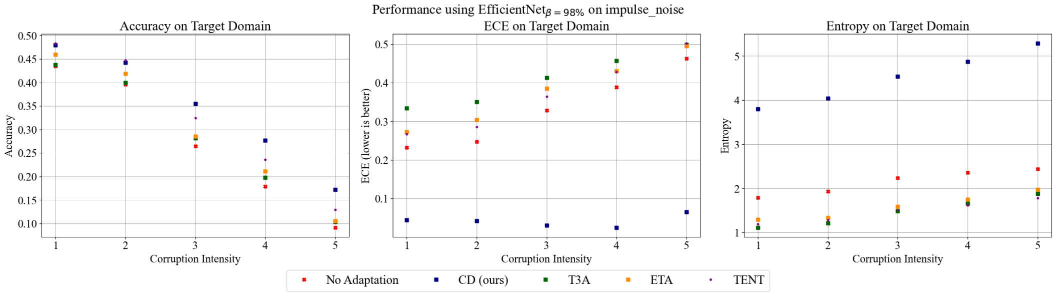

Performance using EfficientNet$_{\beta = 98\%}$ on brightness
Accuracy on Target Domain
Accuracy
0.25
0.30
0.35
0.40
0.45
0.50
0.55
ECE on Target Domain
ECE (lower is better)
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Entropy on Target Domain
Entropy
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Corruption Intensity
1
2
3
4
5
No Adaptation
CD (ours)
T3A
ETA
TENT

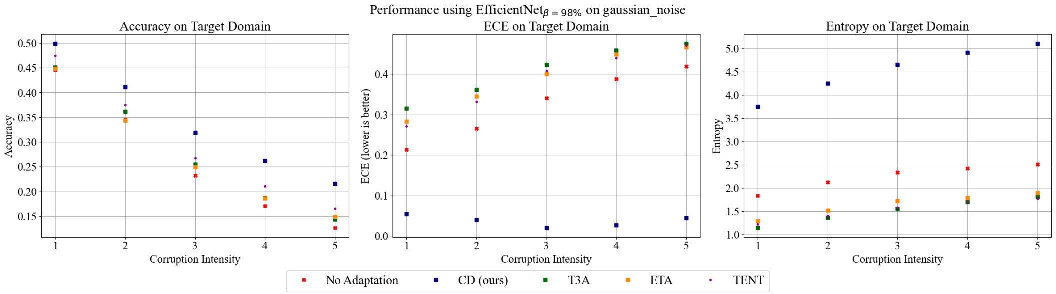

Performance using EfficientNet$_{\beta = 98\%}$ on glass_blur
Accuracy on Target Domain
Accuracy
0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45
ECE on Target Domain
ECE (lower is better)
0.0 0.1 0.2 0.3 0.4 0.5
Entropy on Target Domain
Entropy
2 3 4 5
Corruption Intensity
1 2 3 4 5
No Adaptation
CD (ours)
T3A
ETA
TENT

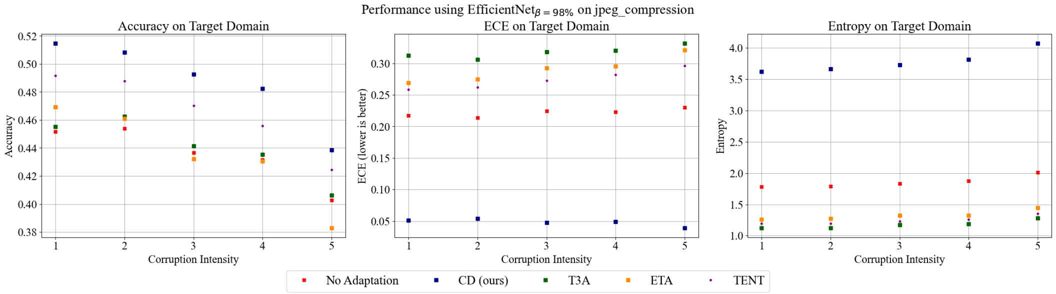

Performance using EfficientNet$_{\beta = 98\%}$ on shot_noise
Accuracy on Target Domain
Accuracy
0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50
ECE on Target Domain
ECE (lower is better)
0.0 0.1 0.2 0.3 0.4
Entropy on Target Domain
Entropy
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
Corruption Intensity
1 2 3 4 5
No Adaptation
CD (ours)
T3A
ETA
TENT

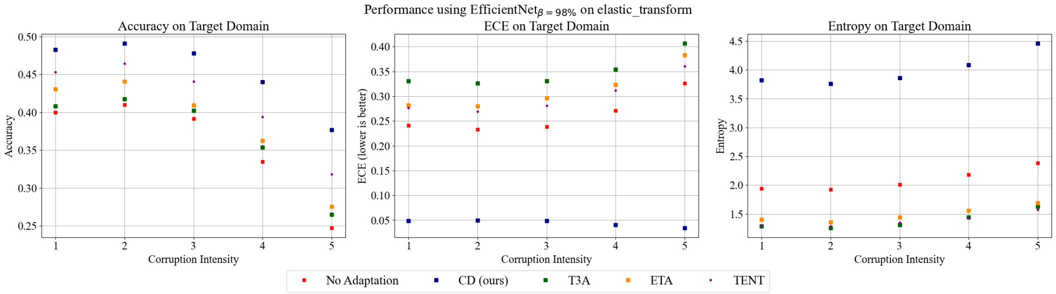

Performance using EfficientNet$_{\beta = 98\%}$ on motion_blur
Accuracy on Target Domain
Accuracy
0.25 0.30 0.35 0.40 0.45 0.50
ECE on Target Domain
ECE (lower is better)
0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
Entropy on Target Domain
Entropy
1.5 2.0 2.5 3.0 3.5 4.0 4.5
Corruption Intensity
1 2 3 4 5
No Adaptation
CD (ours)
T3A
ETA
TENT


Performance using EfficientNet$_{\beta = 98\%}$ on defocus_blur
Accuracy on Target Domain
Accuracy
0.20 0.25 0.30 0.35 0.40 0.45 0.50
ECE on Target Domain
ECE (lower is better)
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
Entropy on Target Domain
Entropy
1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
Corruption Intensity
1 2 3 4 5
No Adaptation
CD (ours)
T3A
ETA
TENT

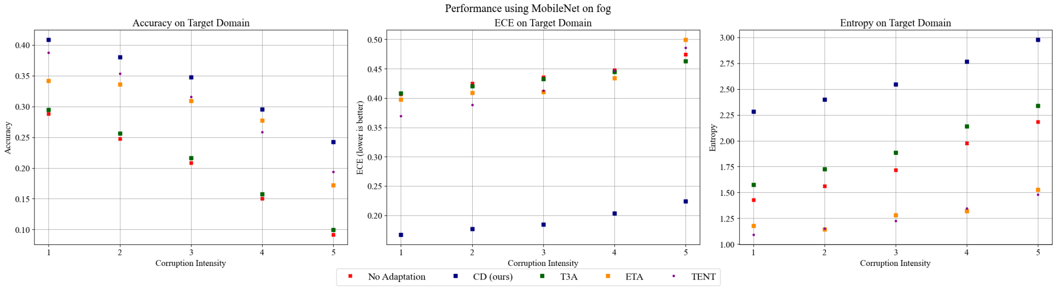

Performance using EfficientNet$_{\beta = 98\%}$ on impulse_noise
Accuracy on Target Domain
Accuracy
0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50
ECE on Target Domain
ECE (lower is better)
0.1 0.2 0.3 0.4 0.5
Entropy on Target Domain
Entropy
1 2 3 4 5
Corruption Intensity
1 2 3 4 5
No Adaptation
CD (ours)
T3A
ETA
TENT

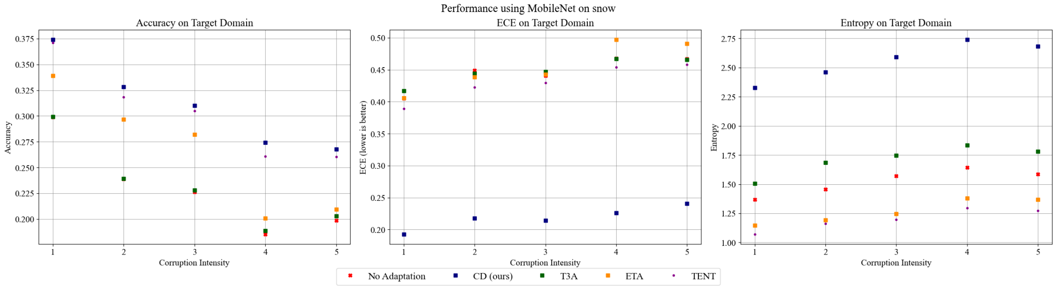

Performance using EfficientNet$_{\beta=98\%}$ on gaussian_noise
Accuracy on Target Domain
Accuracy
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.4
0.3
0.2
0.1
0.0
Entropy on Target Domain
Entropy
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
No Adaptation
CD (ours)
T3A
ETA
TENT

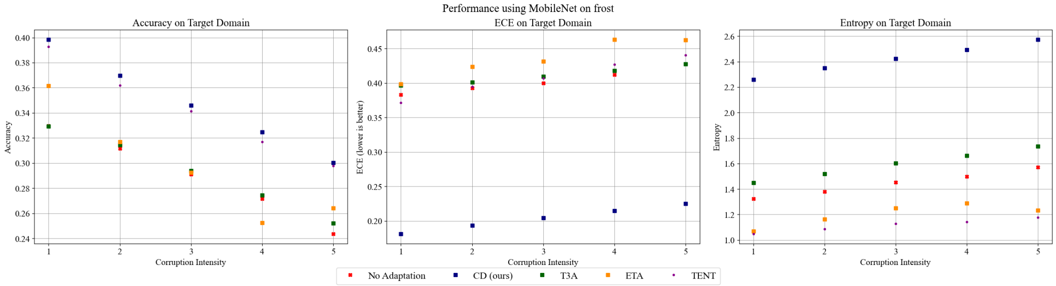

Performance using EfficientNet$_{\beta=98\%}$ on jpeg_compression
Accuracy on Target Domain
Accuracy
0.52
0.50
0.48
0.46
0.44
0.42
0.40
0.38
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.30
0.25
0.20
0.15
0.10
0.05
Entropy on Target Domain
Entropy
4.0
3.5
3.0
2.5
2.0
1.5
1.0
No Adaptation
CD (ours)
T3A
ETA
TENT

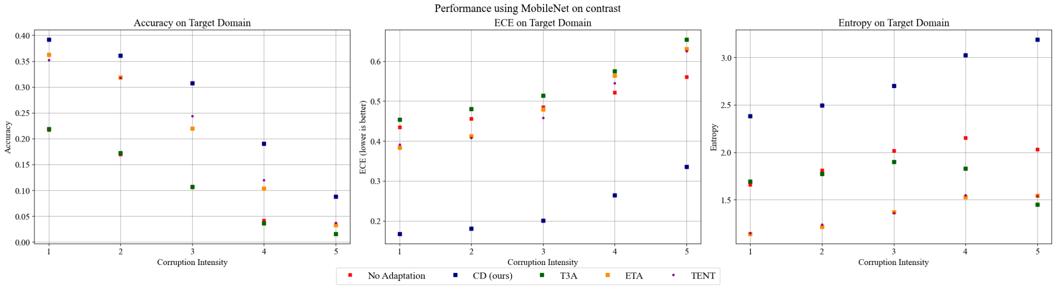

Performance using EfficientNet$_{\beta=98\%}$ on elastic_transform
Accuracy on Target Domain
Accuracy
0.50
0.45
0.40
0.35
0.30
0.25
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
Entropy on Target Domain
Entropy
4.5
4.0
3.5
3.0
2.5
2.0
1.5
No Adaptation
CD (ours)
T3A
ETA
TENT

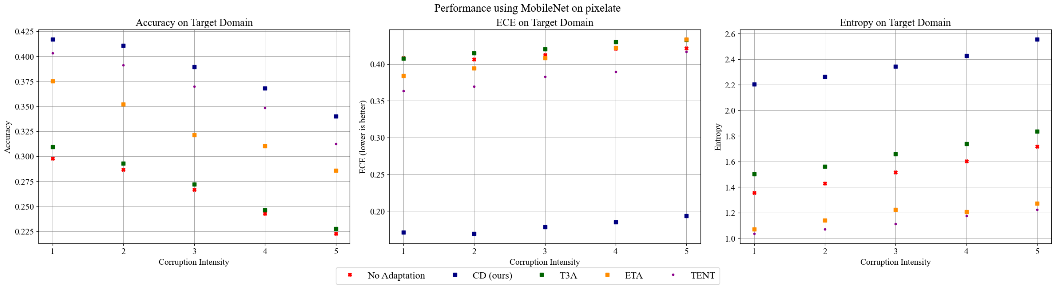
A.3 Results using MobileNet$_{\beta=0\%}$ on TinyImageNet-C

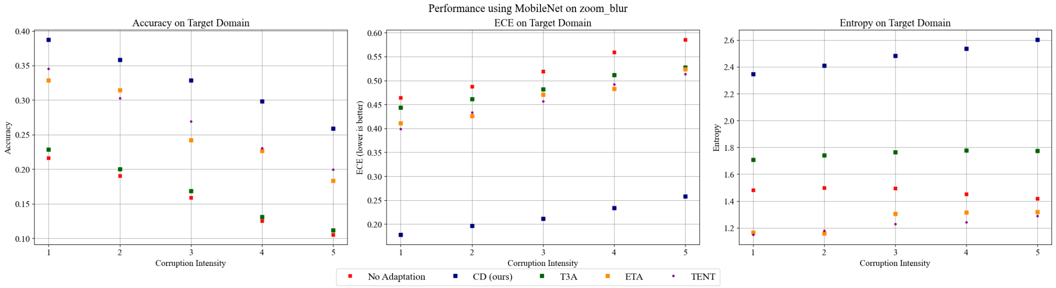


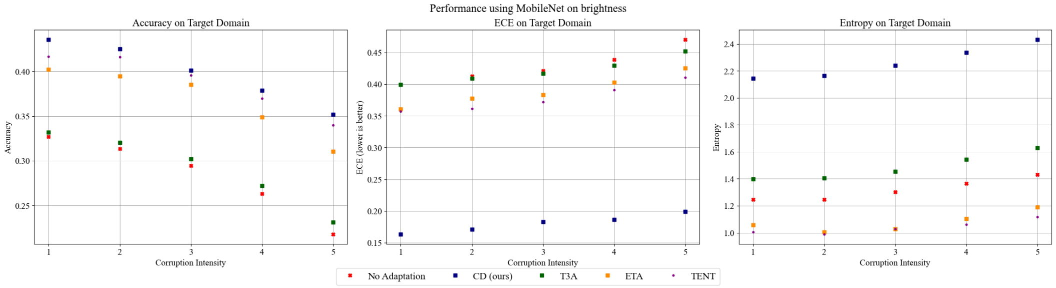


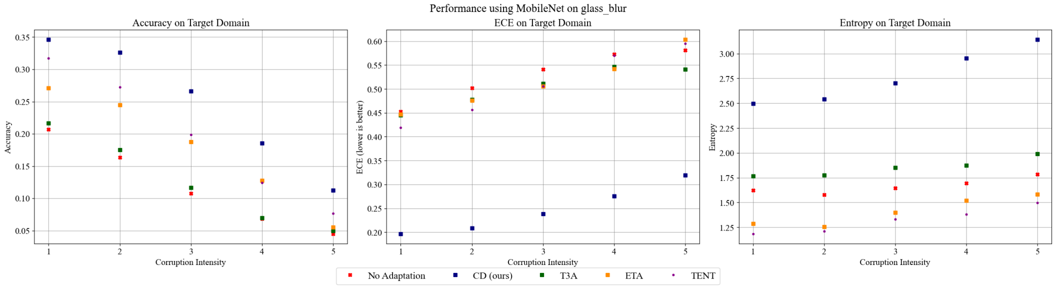


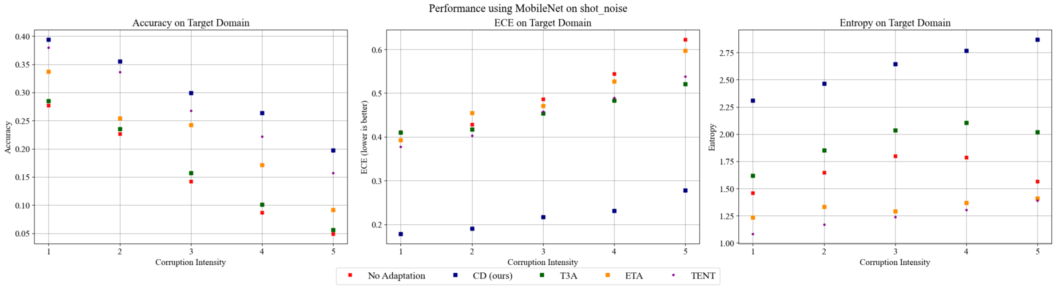

Performance using MobileNet on contrast
Accuracy on Target Domain
Accuracy
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.2
0.3
0.4
0.5
0.6
Corruption Intensity
Entropy on Target Domain
Entropy
1.5
2.0
2.5
3.0
Corruption Intensity
No Adaptation
CD (ours)
T3A
ETA
TENT

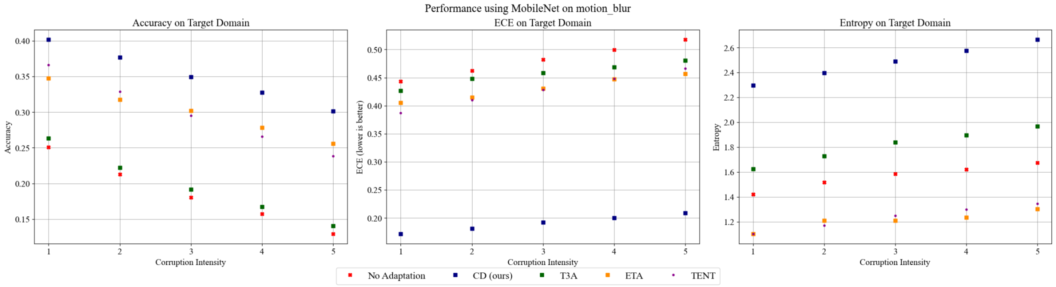

Performance using MobileNet on pixelate
Accuracy on Target Domain
Accuracy
0.225
0.250
0.275
0.300
0.325
0.350
0.375
0.400
0.425
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.20
0.25
0.30
0.35
0.40
Corruption Intensity
Entropy on Target Domain
Entropy
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
Corruption Intensity
No Adaptation
CD (ours)
T3A
ETA
TENT

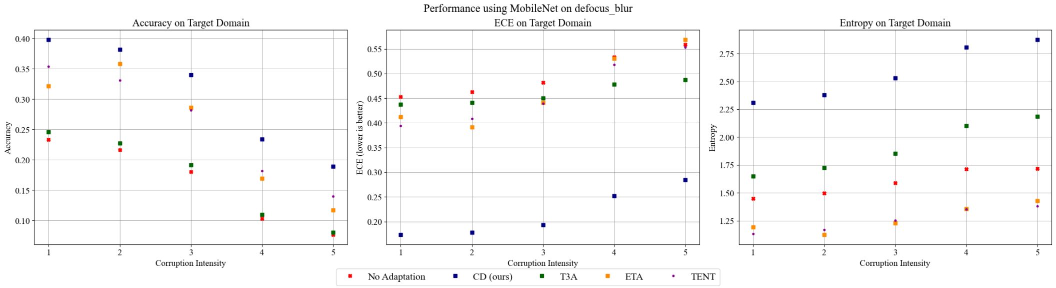

Performance using MobileNet on zoom_blur
Accuracy on Target Domain
Accuracy
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
Corruption Intensity
Entropy on Target Domain
Entropy
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
Corruption Intensity
No Adaptation
CD (ours)
T3A
ETA
TENT

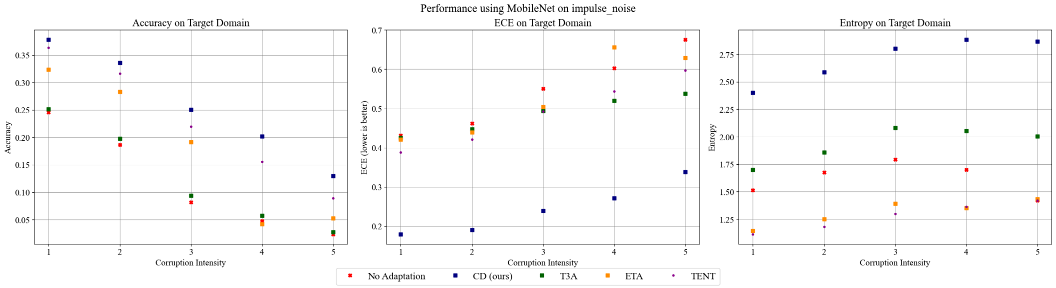

Performance using MobileNet on brightness
Accuracy on Target Domain
Accuracy
0.25
0.30
0.35
0.40
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.15
0.20
0.25
0.30
0.35
0.40
0.45
Corruption Intensity
Entropy on Target Domain
Entropy
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
Corruption Intensity
No Adaptation
CD (ours)
T3A
ETA
TENT

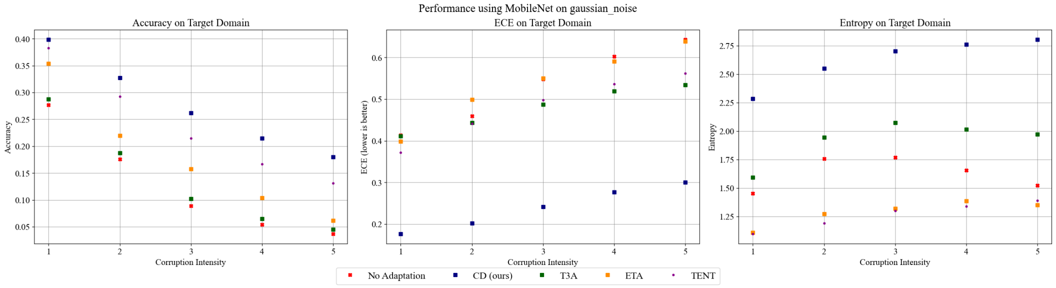

Performance using MobileNet on glass_blur
Accuracy on Target Domain
Accuracy
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
Corruption Intensity
Entropy on Target Domain
Entropy
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Corruption Intensity
No Adaptation
CD (ours)
T3A
ETA
TENT

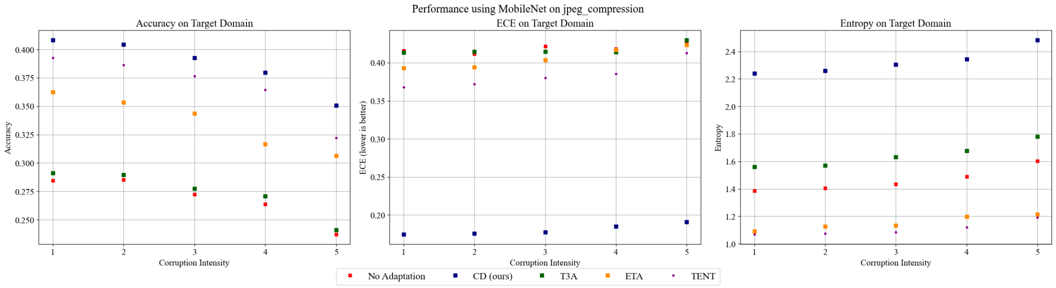

Performance using MobileNet on shot_noise

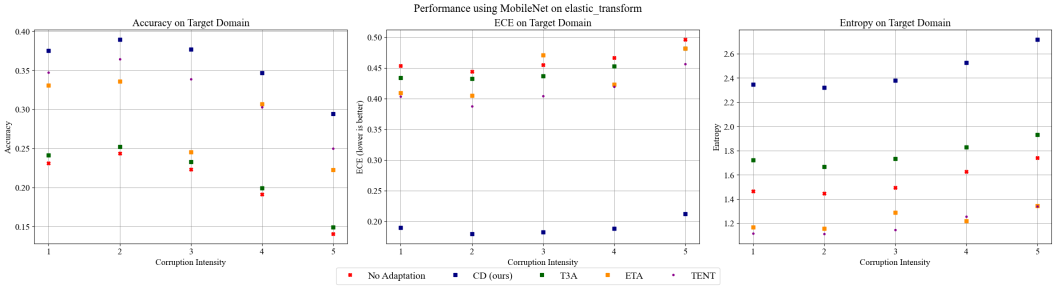

Performance using MobileNet on motion_blur


Performance using MobileNet on defocus_blur


Performance using MobileNet on impulse_noise


Performance using MobileNet on gaussian_noise


Performance using MobileNet on jpeg_compression
Accuracy on Target Domain
Accuracy
0.250
0.275
0.300
0.325
0.350
0.375
0.400
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.20
0.25
0.30
0.35
0.40
Entropy on Target Domain
Entropy
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
No Adaptation
CD (ours)
T3A
ETA
TENT


Performance using MobileNet on elastic_transform
Accuracy on Target Domain
Accuracy
0.15
0.20
0.25
0.30
0.35
0.40
Corruption Intensity
1
2
3
4
5
ECE on Target Domain
ECE (lower is better)
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Entropy on Target Domain
Entropy
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
No Adaptation
CD (ours)
T3A
ETA
TENT

A.4 Other Experiments

Algorithm	Accuracy	ECE	Entropy
No Adaptation	0.6646	0.1979	0.6418
CD$_{\beta=98\%}$	0.6686	**0.0692**	1.5162
CD$_{\beta=0\%}$	0.6720	**0.0622**	1.5576
T3A	**0.6863**	0.2667	0.1857
ETA	0.6778	0.2180	0.4556
TENT	0.6837	0.2105	0.4597

Table 4: Average accuracy, ECE and entropy on the Home Office dataset using the EfficientNet backbone. Domain-to-domain $\sigma^2_{\texttt{max}} = [0.06, 0.03, 1.11]$ for accuracy, ECE and entropy respectively.

Algorithm	Accuracy	ECE	Entropy
No Adaptation	0.5669	0.2424	0.8235
CD	0.5748	**0.06378**	1.8699
T3A	**0.6169**	0.3351	0.1819
ETA	0.5776	0.2395	0.7928
TENT	0.5791	0.2412	0.7761

Table 5: Average accuracy, ECE and entropy on the Home Office dataset using the MobileNet$_{\beta=0\%}$ backbone. Domain-to-domain $\sigma^2_{\texttt{max}} = [0.01, 0.09, 1.11]$ for accuracy, ECE and entropy respectively.

Algorithm	Accuracy	ECE	Entropy
No Adaptation	0.8730	0.1054	0.0812
CD (ours)	0.8794	**0.0789**	0.1565
T3A	**0.8973**	0.0844	0.0645
ETA	0.8827	0.0975	0.0723
TENT	0.8795	0.1007	0.0719

Table 6: Average accuracy, ECE and entropy on the PACS dataset using the EfficientNet$_{\beta=98\%}$ backbone. Domain-to-domain $\sigma^2_{\texttt{max}} = [0.06, 0.03, 0.22]$ for accuracy, ECE and entropy respectively.

A.5 Run-to-Run Variances

- For the Digits dataset: $\sigma^2_{\texttt{max}} = [4.39e-05, 5.51e-04, 3.85e-03]$ for accuracy, ECE and entropy, respectively across all trials using SmallCNN.
- For Home Office using using the EffecientNet$_{\beta=0\%}$ backbone: and $\sigma^2_{\texttt{max}} = [0.05, 1.12e-5, 1.66e-5]$ for entropy and ECE respectively.
- For Home Office using the EffecientNet$_{\beta=98\%}$ backbone: $\sigma^2_{\texttt{max}} = [1.18e-7, 1.46e-5, 1.61e-7]$ for accuracy, ECE and entropy across all trials.
- For Home Office using the MobileNet$_{\beta=0\%}$ backbone: $\sigma^2_{\texttt{max}} = [5.13e-5, 3.81e-5, 9.54e-7]$ for accuracy, ECE and entropy across all trials.
- For the PACS dataset using EfficientNet$_{\beta=98\%}$ backbone: $\sigma^2_{\texttt{max}} = [1.21e-05, 1.02e-13, 2.09e-03]$ for accuracy, ECE and entropy, respectively across all trials.

A.6 SmallCNN

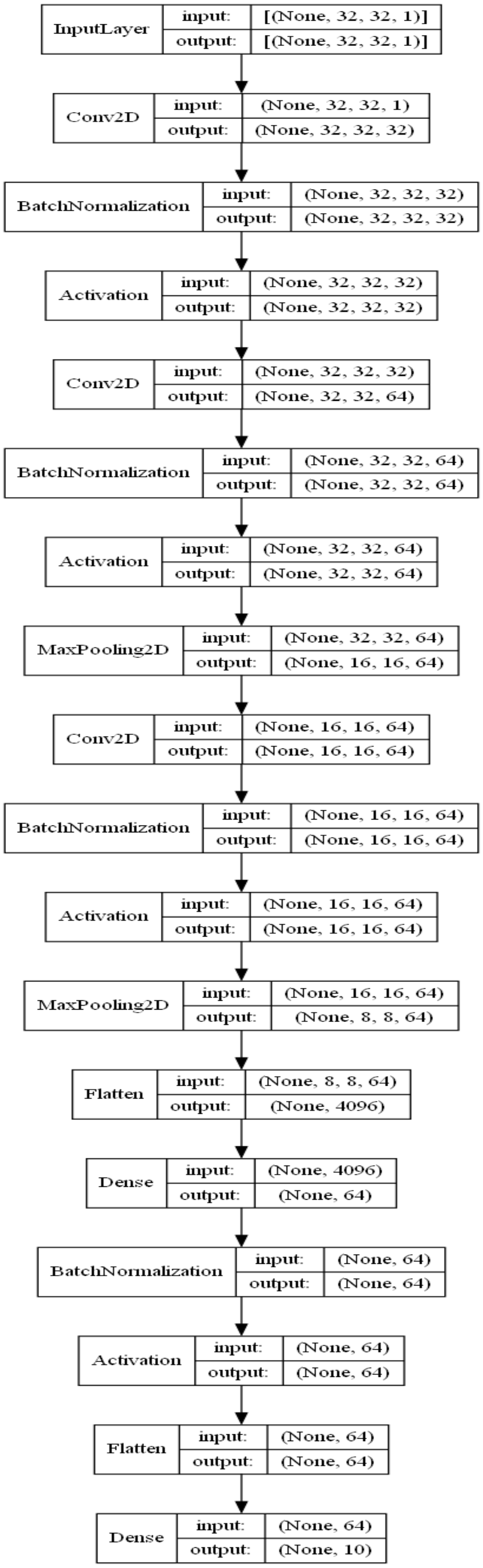

InputLayer
input: [(None, 32, 32, 1)]
output: [(None, 32, 32, 1)]
Conv2D
input: (None, 32, 32, 1)
output: (None, 32, 32, 32)
BatchNormalization
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
Activation
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
Conv2D
input: (None, 32, 32, 32)
output: (None, 32, 32, 64)
BatchNormalization
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
Activation
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
MaxPooling2D
input: (None, 32, 32, 64)
output: (None, 16, 16, 64)
Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 64)
BatchNormalization
input: (None, 16, 16, 64)
output: (None, 16, 16, 64)
Activation
input: (None, 16, 16, 64)
output: (None, 16, 16, 64)
MaxPooling2D
input: (None, 16, 16, 64)
output: (None, 8, 8, 64)
Flatten
input: (None, 8, 8, 64)
output: (None, 4096)
Dense
input: (None, 4096)
output: (None, 64)
BatchNormalization
input: (None, 64)
output: (None, 64)
Activation
input: (None, 64)
output: (None, 64)
Flatten
input: (None, 64)
output: (None, 64)
Dense
input: (None, 64)
output: (None, 10)

Figure 5: SmallCNN's architecture has 319,498 parameters.