A. Mapping between source and target classes

As stated towards the end of Section 3, target classes may belong to a set of *superclasses*. Formally, we note original source classes as $\mathcal{Y}$, and target classes as $\mathcal{Z}$. We require that there exists a mapping $\mu : \mathcal{Y} \to \mathcal{Z} \cup \{\varnothing\}$ that maps each source class to either its unique corresponding superclass in the target domain if it exists, or to the null variable.

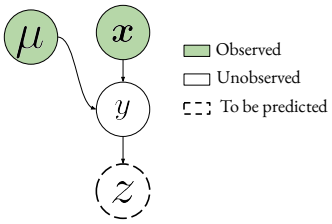

Figure 7. A causal view of the class mapping.

In preliminary experiments, we explored max-pooling following [53]:

$$q_{\boldsymbol{\theta}}(z|\mathbf{x},\mu) = \max_{y:\mu(y)=z} q_{\boldsymbol{\theta}}(y|\mathbf{x}) \tag{6}$$

but found that average pooling performed overall better:

$$q_{\boldsymbol{\theta}}(z|\mathbf{x},\mu) = \frac{1}{|\{y:\mu(y)=z\}|} \sum_{y:\mu(y)=z} q_{\boldsymbol{\theta}}(y|\mathbf{x}) \tag{7}$$

B. Detailed derivation of LAME

We detail the derivation of Eq. (5) from the upper bound (4). Given a solution $\tilde{\mathbf{Z}}^{(n)}$ at iteration n, the goal is find the next iterate $\tilde{\mathbf{Z}}^{(n+1)}$ that minimizes the following constrained problem:

$$\begin{aligned} \min_{\tilde{\mathbf{Z}}} \quad & \sum_{i=1}^{N} \mathrm{KL}(\tilde{\mathbf{z}}_i||\mathbf{q}_i) - \sum_{i=1}^{N}\sum_{j=1}^{N} \tilde{\mathbf{z}}_i^T \tilde{\mathbf{z}}_j^{(n)} \\ \text{s.t} \quad & \tilde{\mathbf{z}}_i^T \mathbb{1}_K,\ \forall i \in \{1,\dots,N\} \end{aligned} \tag{8}$$

The objective function of (8) is strictly convex due to the presence of the KL term, with linear equality constraints. We can write down the associated Lagrangian:

$$\begin{aligned} \mathcal{L}(\tilde{\mathbf{Z}},\boldsymbol{\lambda}) = & \sum_{i=1}^{N} \tilde{\mathbf{z}}_i^T \log(\tilde{\mathbf{z}}_i) - \sum_{i=1}^{N} \tilde{\mathbf{z}}_i^T \log(\mathbf{q}_i) \\ & - \sum_{i=1}^{N}\sum_{j=1}^{N} \tilde{\mathbf{z}}_i^T \tilde{\mathbf{z}}_j^{(n)} + \sum_{i=1}^{N} \boldsymbol{\lambda}_i (\tilde{\mathbf{z}}_i^T \mathbb{1}_K - 1) \end{aligned} \tag{9}$$

where $\boldsymbol{\lambda} = \{\lambda_1,\dots,\lambda_N\}$ represents the vector of Lagrange multipliers associated with the N linear constraints of Problem (8). Let us now compute the derivative of $\mathcal{L}(\tilde{\mathbf{Z}},\boldsymbol{\lambda})$ w.r.t to $\tilde{\mathbf{z}}_i,\ \forall i \in \{1,\dots,N\}$:

$$\nabla_{\tilde{\mathbf{z}}_i} \mathcal{L}(\tilde{\mathbf{Z}},\boldsymbol{\lambda}) = (1+\lambda_i)\mathbb{1}_K + \log(\tilde{\mathbf{z}}_i) - \log(\mathbf{q}_i) - \sum_{j=1}^{N} w_{ij}\tilde{\mathbf{z}}_j^{(n)} \tag{10}$$

By setting the gradients of (10) to 0, we can obtain for the optimal solution $\tilde{\mathbf{z}}_i^{(n+1)}$:

$$\tilde{\mathbf{z}}_i^{(n+1)} = (\mathbf{q}_i \odot \exp(\sum_{j=1}^{N} w_{ij}\tilde{\mathbf{z}}_j^{(n)})) \exp(-(\lambda_i+1)) \tag{11}$$

Combining Eq. (11) with the constraint $\mathbb{1}_K^T \tilde{\mathbf{z}}_i^{(n+1)} = 1$ allows us to recover the Lagrange multiplier:

$$\lambda_i = \log((\mathbf{q}_i \odot \exp(\sum_{j=1}^{N} w_{ij}\tilde{\mathbf{z}}_j^{(n)}))^T \mathbb{1}_K) - 1 \tag{12}$$

Which leads to the final solution:

$$\tilde{\mathbf{z}}_i^{(n+1)} = \frac{\mathbf{q}_i \odot \exp(\sum_{j=1}^{N} w_{ij}\tilde{\mathbf{z}}_j^{(n)}))}{(\mathbf{q}_i \odot \exp(\sum_{j=1}^{N} w_{ij}\tilde{\mathbf{z}}_j^{(n)}))^T \mathbb{1}_K} \tag{13}$$

C. Hyperparameters

Given that the space of hyperparameters grows exponentially in the dimension of the grid, we are forced to make a decision about which subset of hyperparameters to tune, and which one to keep fixed w.r.t. the original methods. Note that, for all NAMs, we adopt the Adam optimizer, as it can also be easily used without any momentum (which was found to be the best choice for non-i.i.d. cases).

We define a common grid-search for all NAMs along the following axes:

Learning rate. The learning rate plays a crucial in the learning dynamic. In particular, too small a learning rate can prevent NAMs for actually improving the model, while too aggressive ones may completely degenerate the model, as shown in Fig. 1. Therefore, we search over three values $\{0.001, 0.01, 0.1\}$.

Optimization momentum. Although not often optimized for, we found the presence of momentum could heavily degrade the performances, especially in non i.i.d. scenarios, as sharp changes in the distribution violate the underlying data distribution smoothness taken by the presence of momentum. We therefore leave the choice between the standard momentum value of 0.9 and no momentum at all 0.

Batch Norm momentum. TENT [56] uses the statistics of the current batch for standardization in Batch Normalization (BN) layers, instead of those from the source distribution computed over the whole source domain. AdaBN [27] method also relies on target samples' statistics

to improve the performances, and so does SHOT [29] (by using the model in the default training mode with a BN momentum of 0.1). While helping in some scenarios, we observed that the use of target statistics in BN normalization procedure can sometimes degrade the results, which echoes the recent findings in [6]. Therefore, we leave methods the choice to only use the statistics from the source domain (`momentum=0`), the statistics from the current batch (`momentum=1`), or a trade-off that allows to update the statistics in a smooth manner (`momentum=0.1`).

Layers to adapt. Following the interesting findings from authors in [6], who showed that adapting only the early layers of a network could greatly help in tackling *prior shifts*, we search over three rough partitions of the set of layers: Adapting the first half of the network while keeping the rest frozen, adapting the second half while keeping the first half frozen, or adapting the full network.

C.1. Hyperparameters for LAME

Given that LAME considers the network fully frozen, all hyperparameters detailed above do not need to be tuned for. Instead, LAME's only hyperparameters are to be found in the choice of the affinity matrix. In this work, we decided to follow a standard choice made in [62] to use a k-NN affinity, where:

$$w(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) = \begin{cases} 1 & \text{if } \phi(\mathbf{x}_i) \in \text{kNN}(\phi(\mathbf{x}_j)) \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

where kNN(.) is the function that returns the set of k nearest neighbours. Therefore, one only needs to tune the value of k, which was selected among $\{1, 3, 5\}$. Note that we tried with other standard kernels, namely the simple linear kernel $w(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ and the radial kernel $w(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) = \exp(-\frac{\|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2}{2\sigma^2})$, with σ chosen as the average distance of each point to its k^{th} neighbour, and k found through validation. A comparison over the 5 model architectures used is provided in Fig. 8, where each vertex indicates the average accuracy over the 7 test scenarios. Overall, LAME equipped with any of the three kernels {kNN, linear, rbf} performs roughly similarly.

D. Cross-shift validation matrices

In Fig. 10, we provide the cross-shift validation matrices for all methods we experimented with. All NAMs suffer from dramatic "off-diagonal degradation" when using a single scenario to tune the hyperparameters. This again highlights that, in order to limit the risk of failure at test time, NAMs need to be evaluated across a broad set of validation scenarios. On the other hand, LAME is much more robust, partly due to the fact that it introduces fewer hyperparameters by design. One extra hyperparameter that we could

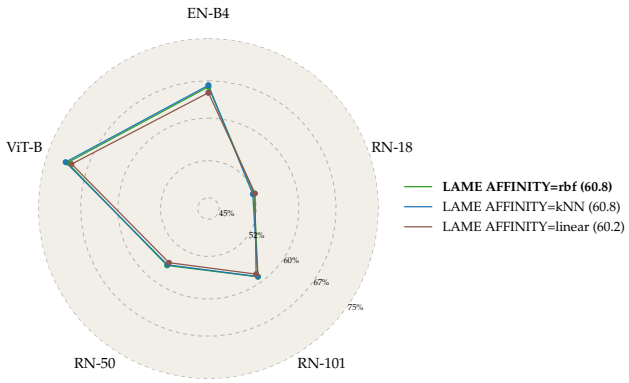

Figure 8. A comparison of three kernels for LAME method. The specific choice of the kernel doesn't seem to have much influence on the performances.

tune is a scalar deciding the relative weight of the two terms in the LAME loss of Eq. (3). We expect that, by tuning this extra hyperparameter, we would achieve overall higher performance for the main experiments, but also slightly worse off-diagonal degradation in the confusion matrix.

E. Ablation study on the batch size

Given that LAME essentially performs output probability correction at the batch level, it is quite important to assess the influence of the size of the batches on the performance of the method. Fig. 9 confirms that LAME preserves a close-to-4% average improvement over the baseline across a wide range of batch sizes.

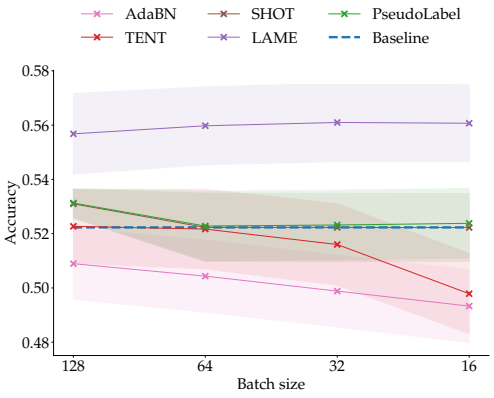

Figure 9. Average accuracy across 7 test scenarios versus batch size, using the Original RN-50. Above 128, NAMs do not fit on a consumer-grade standard 11 GB GPU. In fact, for larger architectures such as ViT-B, a batch size of 16 is already enough to max out memory. Therefore, performing well in the low batch-size regime is a highly desirable property for TTA methods.

Table 1. Summary of the different datasets used.

Used for	Dataset	Short description	# classes	# samples
Validation	ImageNet-Val	The original validation set of ILSVRC 2012 challenge [43].	1000	50'000
	ImageNet-C-Val	Corrupted version of ImageNet-Val, with 9 different corruptions used.	1000	450'000
	ImageNet-C_{16}	Smaller version of ImageNet-C-Val, where only 32 out of the 1000 source classes are mapped to 16 superclasses.	16	14'400
Testing	ImageNet-V2	A recently proposed validation set of ImageNet dataset, collected independently from the original validation set, but following the same protocol. Only 10 samples per class instead of 50 in the original validation set. On ImageNet-V2, models surprisingly drop by $\approx 10\%$ w.r.t. their performance on the original ImageNet-Val.	1000	10'000
	ImageNet-C-Test	Corrupted version of ImageNet-Val, with 10 different corruptions used.	1000	500'000
	ImageNet-Vid	Video dataset corresponding to the validation set of the full ImageNet-Vid [43]. A collection of videos covering diverse situations for factors such as movement type, level of video clutterness, average number of object instance, and several others.	30	176'126
	TAO-LaSOT	LaSOT subset of TAO dataset [11], a popular tracking benchmark, where each sequence comprises various challenges encountered in the wild.	76	6'558

F. Datasets

In Table 1, we present some characteristics of all the datasets used in our experiments. We group the datasets according to whether they are used for the validation or testing stage.

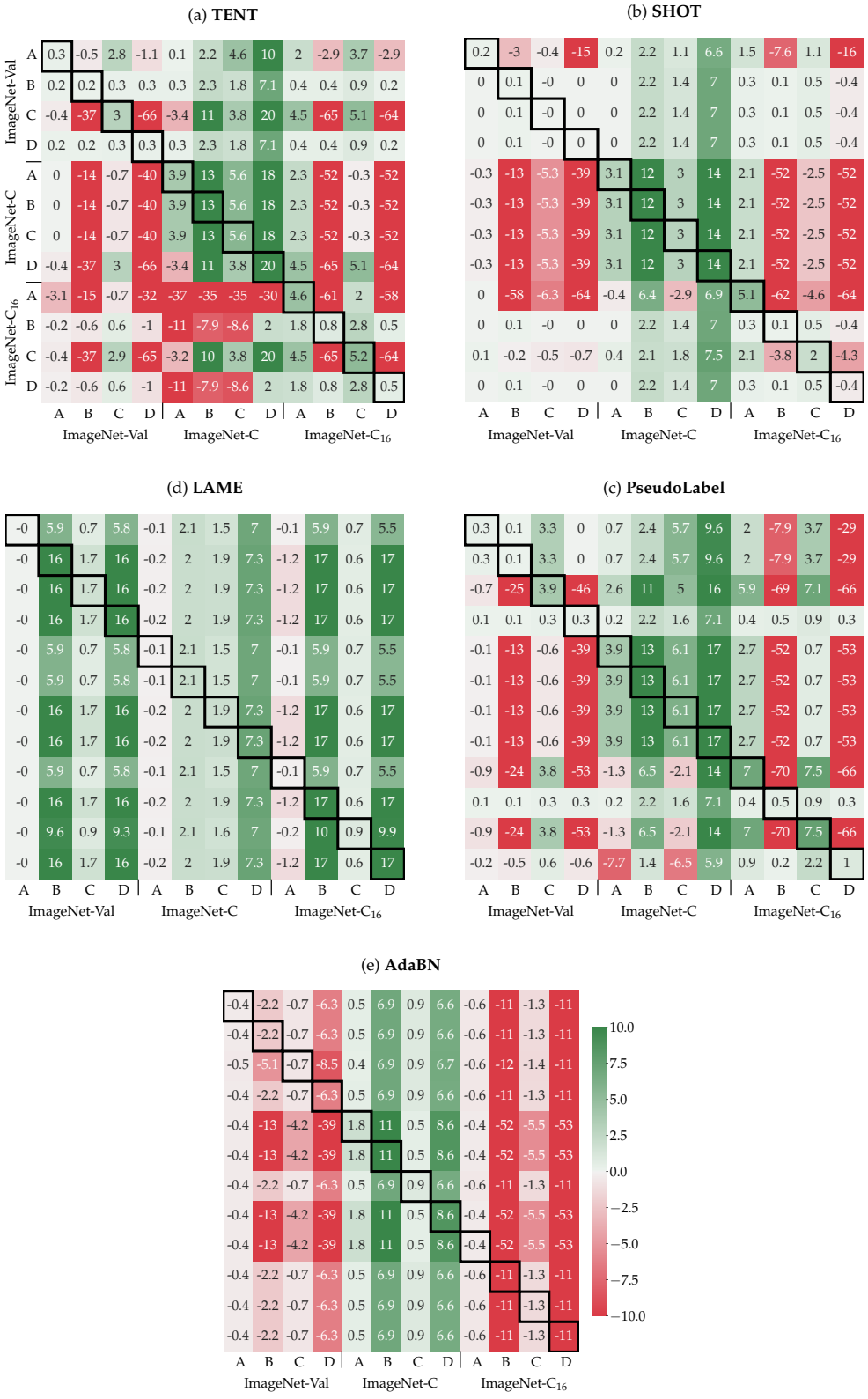

Figure 10. *Cross-shift* validation for all methods. A cell at position (i, j) shows the absolute improvement (or degradation) of the current method w.r.t. to the baseline when using the optimal hyper-parameters for scenario i, but evaluating in scenario j. Legend: A = i.i.d., B = non i.i.d., C = i.i.d. + prior shift, D = non i.i.d. + prior shift. More details on the scenarios in Sec. 6

G. Mapping

Our source model produces output probabilities over all the source classes $\mathcal{Y}$, and one needs to map this output to a valid distribution over target classes $\mathcal{Z}$, as motivated in Section 3. Achieving this, as done in Section A, requires the knowledge of a deterministic mapping μ. We hereby describe how we concretely obtain this mapping in our experiments. Recall that we exclusively rely on ImageNet-trained models, such that $\mathcal{Y}$ always corresponds to the set of ImageNet classes. Therefore, we can directly leverage the existing ImageNet hierarchy in order to design μ. Specifically, μ is defined as follows:

$$\mu(y) = \begin{cases} z & \text{if } \exists\, z \in \mathcal{Z} : y \in Child(z) \\ \varnothing & \text{otherwise} \end{cases} \tag{15}$$

where the function $Child(z)$ outputs the set of all descendants of z in the graph of all ImageNet concepts (or "synsets"). Note that in the case where a conflict exists, *i.e.* class y has multiple target super-classes as parents, we only keep the closest super-class according to the minimum distance in the graph. We qualitative verify that this ancestral scheme produces a sensible mapping of classes between ImageNet and ImageNet-Vid classes in Table 2.

Table 2. Qualitative results obtained with the ancestral mapping scheme for the scenario ImageNet $\rightarrow$ ImageNet-Vid. 279 out of the 1000 original ImageNet were mapped to some superclass.

ImageNet-Vid "superclasses"	ImageNet classes
fox	kit fox, red fox, grey fox, Arctic fox
dog	English setter, Siberian husky, Australian ter...
whale	grey whale, killer whale
red panda	lesser panda
domestic cat	Egyptian cat, Persian cat, tiger cat, ...
antelope	gazelle, impala, hartebeest
elephant	African elephant, Indian elephant
monkey	titi, colobus, guenon, squirrel monkey ...
horse	sorrel
squirrel	fox squirrel
bear	brown bear, ice bear, black bear ...
tiger	tiger
zebra	zebra
sheep	ram
cattle	ox
hamster	hamster
rabbit	Angora, wood rabbit
giant panda	giant panda
lion	lion
airplane	airliner
boat	fireboat, gondola, speedboat, lifeboat ...
bicycle	bicycle-built-for-two, mountain bike
car	ambulance, beach wagon, cab, ...
motorcycle	moped
bird	cock, hen, ostrich, brambling, ...
turtle	loggerhead, leatherback turtle, mud turtle, ...
lizard	banded gecko, common iguana, American chameleo ...
snake	thunder snake, ringneck snake, ...
bus	trolleybus, minibus, school bus
train	bullet train