Appendix

A. Algorithm and implementation details

A.1 Additional motivation and hypothesis

The degradation-restoration process One critical aspect of this work is that the degradation and restoration processes are optimized **jointly instead of individually**. One hypothesis of ours is that LatentDR implements an *implicit adversarial process*: The degradation step D aims to extract features that are (incorrectly) considered as non-discriminative by the classifier g, while the restoration step R learns to recover discriminative components (which might be otherwise ignored due to overfitting). As shown in Table 2 in the paper, our ablations demonstrate that only when D and R are jointly applied, is there a significant enhancement in performance. Additionally, our restoration step R is **guided by the classifier**, which prevents it from being trivial in contrast to a Euclidean distance. We provide more evidence through visualizations as shown in Figure 6(A), the majority of the restored samples (▼) are shifted away from their queries (⋆), and many are out of the training distribution (gray).

Distribution awareness Another crucial aspect of this work is the use of distribution awareness. In our ablation experiments, we show that using a distribution-aware approach to create degraded samples **outperforms unstructured perturbations** generated through additive Gaussian noise. We hypothesize that since LatentDR apply D and R as transformers that consider sample-to-sample relationships, they would alter the gradient flow [18], and help the encoder f to learn better representations. While our experiments and ablations provide strong evidence that suggests that LatentDR can be used as a plug-and-play approach in diverse tasks (domain generalization, long-tail recognition, and medical imaging classification), without strong assumptions or modifications of the sample/label distributions, it remains an interesting problem on which assumption of distribution LatentDR need for it to work.

A.2 Algorithm details

Pseudocode LatentDR can be implemented through the below pseudocode. During the training stage, the degradation operator D and the restoration operator R would create latent augmentations $(z_d, \tilde{y})$ and (z_r, y), correspondingly. The augmented latents would guide the model to learn the relationship across samples, and thus generalize better to unseen sources which span several training domains. During inference, both the degradation and restoration operators are removed.

Algorithm 1: Pseudocode of LatentDR

```
def train(z, g, y, D, R):
    # z:  a batch of latents
    # g:  a classifier
    # y:  one-hot labels of z
    # D: the degradation operator
    # R: the restoration operator
    z_d = D(z)
    z_r = R(z_d, z)
    ỹ = y.sum(0)/y.shape[0]
    l1 = F.cross_entropy(g(z), y)
    l2 = F.cross_entropy(g(z_d), ỹ)
    l3 = F.cross_entropy(g(z_r), y)
    loss = l1 + l2 + l3
    loss.backward()
_
def pred(z, g):
    # The prediction stage
    return g(z)
```

Algorithm variants For the degradation and restoration operators D and R, we tested two simple variants of the transformer layer. The formulation of the two variants are denoted as below:

$$\begin{aligned} Z'_{\ell+1} &= \mathrm{LN}(Z_\ell + \mathrm{AttN}(Z_\ell)) \\ Z_{\ell+1} &= \mathrm{LN}(Z'_{\ell+1} + \mathrm{FF}(Z'_{\ell+1})),\ 0 \le \ell \le L-1 \end{aligned} \tag{13}$$

$$\begin{aligned} Z'_{\ell+1} &= \mathrm{LN}(Z_\ell) + \mathrm{AttN}(Z_\ell) \\ Z_{\ell+1} &= \mathrm{LN}(Z'_{\ell+1}) + \mathrm{FF}(Z'_{\ell+1}),\ 0 \le \ell \le L-1 \end{aligned} \tag{14}$$

where AttN denotes either the self-attention operation (MSA($\cdot$)) or the pooling operation (Pool($\Omega(\cdot)$)) for LatentDR (SA) or LatentDR (Pool). In equation 13 and 14, the layerwise normalization (LN) is applied later than or prior to the attention (AttN) and feedforward (FF) operations, respectively. We used the equation 13 variant for our experiments in DomainBed and medical imaging classification tasks, where we did not observe a significant difference between the two variants. In long-tail recognition experiments, we used the equation 14 variant, where we observed that it outperforms the other variant by a large margin.

A.3 Implementation details

In all our experiments, we used a one-layer Transformer encoder for both the degradation process and the restoration process for simplicity.

DomainBed experiments For all of the benchmark models, we follow the training and evaluation protocol as in

[10], where we used their default algorithm-agnostic hyperparameters including batch size, learning rate, dropout, and weight decay. Due to the large loss value of `LatentDR`, we performed a fixed learning rate adjustment of 50% for all of our experiments, and fixed the other hyperparameters. `LatentDR` also contains many algorithm-specific parameters, including the dimensionality of the transformers (both the attention head dimension $\mathrm{dim\text{-}head}$ and the feed-forward dimension $\mathrm{dim\text{-}ff}$), and transformer dropout rate. Searching all hyperparameters on each dataset would require heavy computational resources. Thus, we search the algorithm-specific hyperparameters on the PACS datasets, and use the same hyperparameters on all other datasets.

For `LatentDR` (SA), we search $\mathrm{dim\text{-}head}$ and $\mathrm{dim\text{-}ff}$ in $[\mathrm{dim}, \mathrm{dim}/2, \mathrm{dim}/4, \mathrm{dim}/8]$, where dim is the latent space dimensionality. The transformer dropout rate is searched in $20\%, 50\%, 70\%$. We used $\mathrm{dim}/4$ for both dimensions, and used 50% dropout rate after the parameter selection. Our competitor model BatchFormer [18] is defaulted to select dim for both dimensions and a dropout rate of 50% as in their paper. To ensure the fairness of comparison, we performed the same hyperparameter search for BatchFormer and used their default values as it returns the best results.

For `LatentDR` (Pool), we search the hyperparameter $\mathrm{dim\text{-}head}$ in $[\mathrm{dim}/8, \mathrm{dim}/16, \mathrm{dim}/32]$ and the hyperparameter $\mathrm{dim\text{-}ff}$ in $[\mathrm{dim}/4, \mathrm{dim}/8]$. The dropout rate is fixed to be 50% due to the search results in the previous model to reduce additional computational costs. We used $\mathrm{dim\text{-}head} = \mathrm{dim}/32$, $\mathrm{dim\text{-}ff} = \mathrm{dim}/8$ after the parameter selection. Interestingly, we noticed that replacing the self-attention operator with a pooling operator requires fewer additional parameters in training, while the additional stochasticity ensures the robustness of the performance.

Medical imaging experiments For the first four datasets, we follow the training pipeline and dataset splits in [61], where we used a ResNet-18 for 2D images and a ResNet-18-based 3D backbone to benchmark the performance. The ResNet-18-based 3D backbone is built with the ACSConv package. All models are trained for 100 epochs till convergence. For Camelyon17, we followed the model, data, and training setup in [48] and used a DenseNet-121 backbone. We used an SGD optimizer with lr=0.0001 and momentum=0.9. All models are trained for 20 epochs with a batch size=32. We applied the same set of hyperparameters based on our experiments in DomainBed, which demonstrated the robustness of our approach as it is insensitive to the selection of hyperparameters.

Long-tail recognition experiments Inspired by the robust performance of [18] on long-tail recognition tasks, we applied our method on CIFAR-100-LT. We add `LatentDR` (SA) on top of BCL and BALMS following their default hyperparameter and training settings. For our model, we used the same hyperparameters as above, and used the eq.14 variant of transformer formulation to achieve the best performance in long-tail recognition tasks.

B. Full results on DomainBed

We provide the full results for all augmentation methods in all tested DomainBed datasets as below. Note that the details of the break-down performance for other algorithms can be found in [10] and [43].

PACS The breakdown performances are shown in Table 6, where σ is the standard deviation (SD).

	A	C	P	S	Avg.
ERM	84.9	80.6	95.9	75.0	84.1
+ Mixup	84.2	79.4	96.0	72.8	83.1
+ CutMix	81.5	75.8	95.6	68.9	80.4
+ Manifold-Mixup	84.9	79.5	96.7	75.8	84.2
+ MixStyle†	86.8	79.0	96.6	78.5	85.2
+ BatchFormer	82.6	79.3	95.4	73.5	82.7
+ LatentDR (SA)	87.4	81.2	97.8	76.7	$\underline{85.8}$
SD ($\pm\sigma$)	1.3	1.3	0.2	1.8	0.8
+ LatentDR (Pool)	86.3	82.6	97.1	79.2	**86.3**
SD ($\pm\sigma$)	1.1	1.4	0.4	1.7	0.9

Table 6. Full results on PACS.

VLCS The breakdown performances are shown in Table 7, where σ is the standard deviation (SD).

	C	L	S	V	Avg.
ERM	96.2	63.9	72.2	74.4	76.7
+ Mixup	97.7	64.8	70.9	74.2	76.9
+ CutMix	95.8	62.7	71.0	70.0	74.9
+ Manifold-Mixup	98.4	62.1	75.1	75.7	77.8
+ MixStyle†	98.6	64.5	72.6	75.7	77.9
+ BatchFormer	97.0	64.5	70.9	74.5	76.7
+ LatentDR (SA)	97.8	64.5	73.9	78.4	**78.7**
SD ($\pm\sigma$)	1.0	0.9	1.6	1.6	0.7
+ LatentDR (Pool)	98.0	66.2	69.4	78.4	$\underline{78.0}$
SD ($\pm\sigma$)	0.6	1.2	1.0	1.7	0.5

Table 7. Full results on VLCS.

Office-Home The breakdown performances are shown in Table 8, where σ is the standard deviation (SD).

TerraIncognita The breakdown performances are shown in Table 9, where σ is the standard deviation (SD).

	A	C	P	R	Avg.
ERM	58.8	52.0	73.3	75.1	64.8
+ Mixup	61.8	53.3	75.6	77.2	67.0
+ CutMix	61.2	52.7	77.0	76.6	66.9
+ Manifold-Mixup	63.0	54.4	75.4	76.8	67.4
+ MixStyle†	51.1	53.2	68.2	69.2	60.4
+ BatchFormer	60.0	51.9	74.5	76.7	65.8
+ LatentDR (SA)	65.3	54.9	77.3	78.5	**69.0**
SD ($\pm\sigma$)	0.5	1.0	0.7	0.5	0.3
+ LatentDR (Pool)	63.6	56.1	75.6	78.2	<u>68.4</u>
SD ($\pm\sigma$)	0.5	0.4	0.8	0.6	0.3

Table 8. Full results on Office-Home.

	L100	L38	L43	L46	Avg.
ERM	54.2	41.0	55.4	37.5	47.0
+ Mixup	59.8	41.9	56.3	33.9	48.0
+ CutMix	63.5	50.1	60.5	34.3	**52.1**
+ Manifold-Mixup	57.4	41.4	55.3	31.1	46.3
+ MixStyle†	54.3	34.1	55.9	31.7	44.0
+ BatchFormer	54.0	43.4	56.0	41.0	48.6
+ LatentDR (SA)	49.6	47.1	58.2	44.2	<u>49.8</u>
SD ($\pm\sigma$)	2.6	2.8	1.1	2.5	1.5
+ LatentDR (Pool)	57.6	46.8	58.8	34.8	49.5
SD ($\pm\sigma$)	3.6	3.0	1.0	2.2	1.9

Table 9. Full results on TerraIncognita.

DomainNet The breakdown performances are shown in Table 10.

C. Additional ablations

Sharing classifier weights. Our model trains a classifier g on top of the encoder f, and uses the same classifier g to regularize the model to learn degraded samples and restored samples. Following [18], we tested whether training two different classifiers, one for the original loss, and the other for the corruption/restoration steps, would benefit the training. With a separate classifier, our model gets an average of 83.7% accuracy on PACS, an over 2% decrease. It seems it is critical to share classifier weights for our method. We hypothesize that this is because BatchFormer relies on the gradient flow across samples to guide learning, while `LatentDR` uses latent augmentations to regularize training.

Increasing the batch size. To understand if our method would benefit from learning with larger batch size, we evaluate the performance of `LatentDR` on the PACS dataset with $B = [4, 8, 16, 32, 64, 100]$, where each training domain provides B samples for training (thus, the batch size is $3B$ for the PACS dataset). As shown in Figure 5, training `LatentDR` requires a sufficiently large batch size to capture

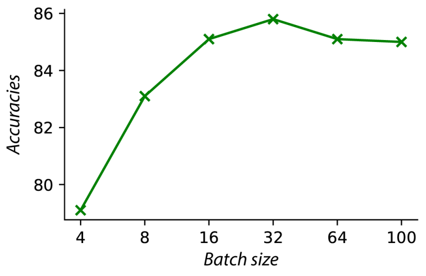

Figure 5. *Ablation on batch size.* We evaluate the performance of `LatentDR` (SA) with different batch size on the PACS dataset.

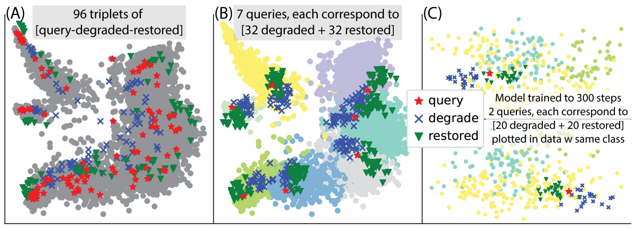

Figure 6. Latent queries ($\star$), the degraded ($\times$), and the restored ($\blacktriangledown$). The left two figures show all training data as background.

rich and meaningful sample-to-sample relationship across different classes and domains. However, further increasing the batch size does not further increase the performance of our method. We hypothesize that it might be challenging for the small-scale transformer to capture information across too many tokens inside the same batch.

D. Additional visualizations

Additional details and copies of Figure 1 We provide additional visualizations as in Figure 6 as additional evidence to Figure 1. Figure 6(A) shows a random batch (96 samples) of queries ($\star$), their degraded pairs ($\times$), and their restored pairs ($\blacktriangledown$) in one forward pass. We note that the restored latents typically are **far away from the original latents**. In Figure 6(B), we show how we generated Figure 1(B): For each random query ($\star$) selected, we insert it into 32 random batches from the training data, and plot both their degraded and restored latents on top of the original representations. In Figure 6(C), we generate Figure 1(A) using a model that is trained on PACS for 300 steps, with training data from the same class as the background.

Visualizations on the testing domain We provide the direct visualization of latent space as in Figure 7, where we use a T-SNE to visualize the model's latent on the testing domain (domain A) for models that are trained on the PACS dataset. Different color represents different classes for the classification task on PACS. As shown in Figure 7,

	clipart	infograph	painting	quickdraw	real	sketch	Avg.
ERM	60.4	19.2	48.1	12.4	60.4	50.9	41.9
+ Mixup	62.1	20.9	49.1	14.5	60.2	51.2	43.0
+ CutMix	61.7	20.6	49.6	13.7	61.9	51.6	43.2
+ Manifold-Mixup	62.4	20.8	49.1	13.4	60.7	51.5	43.0
+ MixStyle†	51.9	13.3	37.0	12.3	46.1	43.4	34.0
+ BatchFormer	62.6	19.4	48.5	13.2	61.8	51.6	42.8
+ LatentDR (SA)	63.6	22.3	51.5	14.6	64.7	54.0	**45.1**
+ LatentDR (Pool)	61.1	21.1	50.6	15.3	62.8	52.5	<u>43.9</u>

Table 10. Full results on DomainNet.

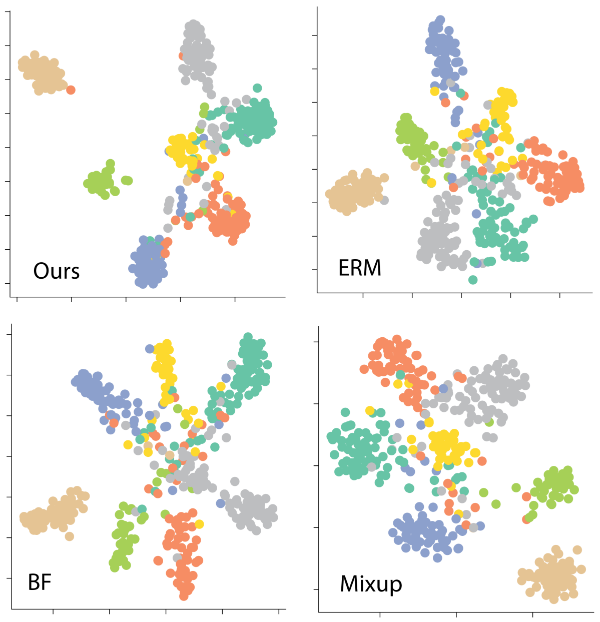

Figure 7. *Latent space direct visualization.* We visualize the latent spaces for ERM, Mixup [64], BatchFormer [18] (BF), and `LatentDR` (Ours-SA) using a T-SNE on the testing domain. Our model provides the best alignment (closeness of latents from the same class) and the best uniformity (most information from the training data is preserved).

`LatentDR` provides the most clustered latents for each class in comparison to other methods, while for each class, `LatentDR` provides clusters with the most circular shape. These properties are further demonstrated through the measurement of alignment and uniformity, where our method achieves the best 'closeness' score for classification, and the best latent 'diversity' score.