A Appendix / supplemental material

A.1 Experimental setup

Data. We use the CIFAR-10 and CIFAR-10C datasets. CIFAR-10C is used to evaluate model robustness, and extends CIFAR-10 by algorithmically applying 15 types of common corruptions at 5 levels of severity (1-5), resulting in 75 distinct test sets. Each type of corruption belongs to one of four categories: noise patterns, blurs, weather effects and digital transformations [29].

Model. Throughout this work, all our DNN experiments use ResNet [30]. Specifically, we use a publicly-available pretrained ResNet18 obtained from the Huggingface model repository [31] which was trained on CIFAR-10 [32] for 300 epochs using SGD optimizer implemented as `torch.optim.SGD(lr=0.1, momentum=0.9, weight_decay=0.0005, nesterov=True)` and a ReduceLROnPlateau scheduler.

Fine-tuning. When fine-tuning ResNet18 + AnchorBlock, we train the combined model on the CIFAR-10 training set for 50 epochs using an SGD optimizer implemented as `torch.optim.SGD(lr=0.01, momentum=0.5, weight_decay=0.01)`. Every 5 epochs, we update the weights of the class covariance heads by re-computing the effective class covariance matrices using the latest training representations.

Computing task-relevant class covariance. To compute task-relevant class covariance, we first run the model on the entire CIFAR-10 training set to obtain representations, which we extract from the last linear layer (dimensionality 512). These representations are in the form $\mathbf{H}_{\text{train}} \in \mathbb{R}^{P\times M\times N}$, where P is the number of classes, M is the number of points (or samples) per class, and N is the dimensionality. Then we follow the standard procedure for computing anchor points described in [6]. This process involves Gaussian sampling of K normal vectors and calculating the corresponding anchor points for each sample by solving a quadratic programming problem. In our experiments, we set `K = 200`. The process returns K anchor points for each manifold, $\mathbf{S} \in \mathbb{R}^{P\times K\times N}$. Per Eq. (3), then for each class we compute the task-relevant class covariance matrix and stack them, which returns the task-relevant class covariance tensor $\mathbf{C}^{\text{eff}} \in \mathbb{R}^{P\times N\times N}$.

Robustness metrics. Per the convention in [33], for each corruption type we track the *Corruption Error* (CE), which is the classification error rate, $1-$accuracy. To assess overall corruption robustness, we use *Mean Corruption Error*(mCE) which averages corruption error over all 75 corruption types. We also report *Clean Error*, which is the classification error rate on the original CIFAR-10 test set.

A.2 Augmentation details

This section provides the detailed implementation on the augmentation used in Section 3.1, where we perform a controlled representation manipulation on the model's CIFAR-10C representations to test our intuitions about effective geometry. Let $H_\mu^i \in \mathbb{R}^N$ be the representation of the i-th sample of the μ-th class. We run the pretrained model on CIFAR-10C and extract activations $\mathbf{H} \in \mathbb{R}^{P\times M\times N}$. Then, we scale each H_μ^i along the eigenspace of the μ-th effective class covariance matrix (see Algorithm 1). For our control, we scale each H_μ^i along the eigenspace of the μ-th point-cloud covariance matrix (see Algorithm 2)

Algorithm 1: Representation augmentation using task-relevant class covariance

Data: $\mathbf{H} \in \mathbb{R}^{P\times M\times N}$, $\mathbf{S} \in \mathbb{R}^{P\times K\times N}$
Result: $\mathbf{H}^{\text{Aug}} \in \mathbb{R}^{P\times M\times N}$; `/* Augmented representation */`
for $\mu \leftarrow 1$ **to** P **do**
  $\mathbf{C}_\mu \leftarrow \text{Cov}(\mathbf{S}_\mu)$; `/* Covariance matrix of task-relevant class manifold */`
  $\mathbf{H}_\mu^{\text{Aug}} \leftarrow \mathbf{H}_\mu \mathbf{C}_\mu$; `/* Scale` $\mathbf{H}_\mu$ `along eigenvectors of covariance */`
  $\mathbf{H}_\mu^{\text{Aug}} \leftarrow \frac{\|\mathbf{H}_\mu\|_F}{\|\mathbf{H}_\mu^{\text{Aug}}\|_F}\mathbf{H}_\mu^{\text{Aug}}$; `/* Normalize to Frobenius norm of` $\mathbf{H}_\mu$ `*/`

Algorithm 2: Representation augmentation using point-cloud class covariance

Data: $\mathbf{H} \in \mathbb{R}^{P \times M_{\text{test}} \times N}$, $\mathbf{H}_{\mu}^{\text{train}} \in \mathbb{R}^{P \times M_{\text{train}} \times N}$

Result: $\mathbf{H}^{\text{Aug}} \in \mathbb{R}^{P \times M \times N}$; /* Augmented representation */

for $\mu \leftarrow 1$ **to** P **do**

$\mathbf{C}_{\mu} \leftarrow \text{Cov}(\mathbf{H}_{\mu}^{\text{train}})$; /* Covariance matrix of point-cloud class manifold */

$\mathbf{H}_{\mu}^{\text{Aug}} \leftarrow \mathbf{H}_{\mu}\mathbf{C}_{\mu}$; /* Scale $\mathbf{H}_{\mu}$ along eigenvectors of covariance */

$\mathbf{H}_{\mu}^{\text{Aug}} \leftarrow \frac{\|\mathbf{H}_{\mu}\|_F}{\|\mathbf{H}_{\mu}^{\text{Aug}}\|_F}\mathbf{H}_{\mu}^{\text{Aug}}$; /* Normalize to Frobenius norm of $\mathbf{H}_{\mu}$ */

A.3 AnchorBlock architecture details

In our setup, AnchorBlock replaces the final readout layer of a deep classifier. For a P-class classification problem, AnchorBlock consists of P parallel "class covariance" heads followed by a shared linear readout layer. Each head is represented by a function $f_i : \mathbb{R}^N \to \mathbb{R}^N$ for $i = [P]$. In the forward pass, the representation vector $h \in \mathbb{R}^N$ is simultaneously processed by all heads. The resulting augmented representation vectors $\tilde{h}_1, \tilde{h}_2, ..., \tilde{h}_P$ are simultaneously passed through the shared readout layer parameterized by $W \in \mathbb{R}^{N \times P}$ to each produce vectors: $z_i = W^T \tilde{h}_i, z_i \in \mathbb{R}^P$. The P vectors are then concatenated to form matrix $Z = [z_1, z_2, ..., z_P], Z \in \mathbb{R}^{P \times P}$. Finally, the diagonal elements of Z are extracted to form the final prediction vector $\hat{y} = \text{softmax}(\text{diag}(Z)) \in \mathbb{R}^P$.

A.4 Additional AnchorBlock results

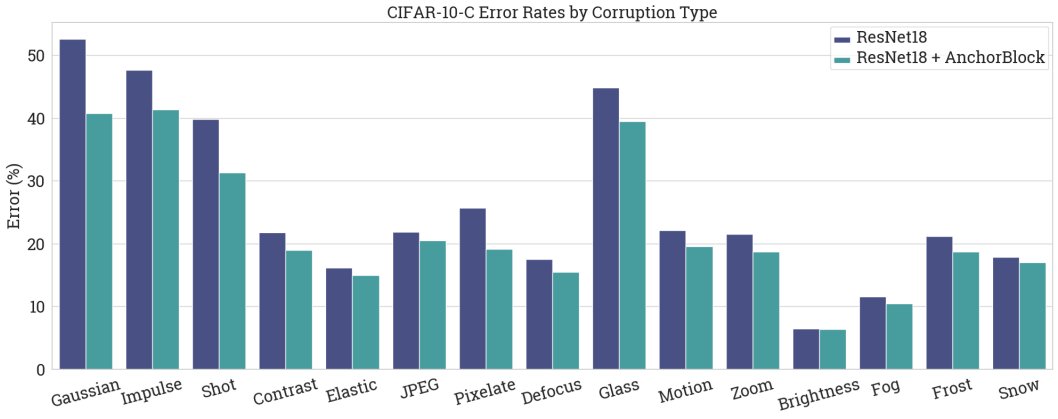

Figure 4: Comparison of mean corruption error between ResNet18+AnchorBlock and ResNet18.