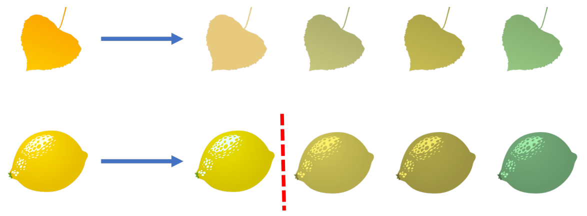
Learning Instance–Specific Augmentations by Capturing Local Invariances

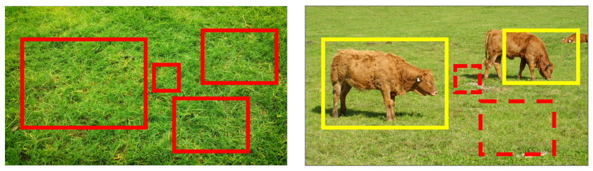
Ning Miao [1] **Tom Rainforth** [1] **Emile Mathieu** [1] **Yann Dubois** [2] **Yee Whye Teh** [1] **Adam Foster** [3] **Hyunjik Kim** [4]

Abstract

We introduce *InstaAug*, a method for automatically learning input-specific augmentations from data. Previous methods for learning augmentations have typically assumed independence between the original input and the transformation applied to that input. This can be highly restrictive, as the invariances we hope our augmentation will capture are themselves often highly input dependent. InstaAug instead introduces a learnable *invariance module* that maps from inputs to tailored transformation parameters, allowing local invariances to be captured. This can be simultaneously trained alongside the downstream model in a fully end-to-end manner, or separately learned for a pre-trained model. We empirically demonstrate that InstaAug learns meaningful input-dependent augmentations for a wide range of transformation classes, which in turn provides better performance on both supervised and self-supervised tasks.



(a) Color jittering (red line indicates class boundary)



(b) Cropping

Figure 1: Different inputs require different augmentations. In (a), a leaf is invariant to color change from yellow to green, but the same transformation changes a lemon to a lime. In (b), the same effect is shown for cropping. Solid rectangles represent the patches that preserve the labels of the original images ([left] grass, [right] cattle), while dashed rectangles represent patches with different labels.

1. Introduction

Data augmentation is an important tool in deep learning (Shorten & Khoshgoftaar, 2019). It allows one to incorporate inductive biases and invariances into models (Chen et al., 2019; Lyle et al., 2020), providing an effective regularization technique that aids generalization (Goodfellow et al., 2016). It has proved particularly successful for computer vision tasks, forming an essential component of many modern supervised (Krizhevsky et al., 2012; Perez & Wang, 2017; Mikołajczyk & Grochowski, 2018; Cubuk et al., 2020) and self-supervised (Bachman et al., 2019; Chen et al., 2020; Tian et al., 2020; Foster et al., 2021) approaches.

Algorithmically, data augmentations apply a *random transformation* $\tau : \mathcal{X} \rightarrow \mathcal{X}$, $\tau \sim p(\tau)$, to each input data point $\mathbf{x} \in \mathcal{X}$, before feeding this *augmented* data into the downstream model. These transformations are resampled each time the data point is used (e.g. at each training epoch), effectively populating the training set with additional samples. Augmentation is also sometimes used at test time by ensembling predictions from multiple transformations of the input. A particular augmentation is defined by the choice of the *transformation distribution* $p(\tau)$, whose construction forms the key design choice. Good transformation distributions induce substantial and wide-ranging changes to the input, while preserving the information needed for prediction.

To try and ensure a good augmentation scheme, previous work has looked to *learn* this transformation distribution from data (Cubuk et al., 2018; Lim et al., 2019; Benton et al., 2020). However, existing approaches typically assume independence between the input $\mathbf{x}$ and the transformation distribution $p(\tau)$. As such, they are only able to learn *global invariances*, severely limiting their flexibility and potential impact. For example, when using color jittering, changing the color of a leaf from yellow to green would preserve its label, but the same transformation would change a lemon to a lime (see Figure 1a). This transformation cannot, therefore,

[1]Dept. of Statistics, University of Oxford [2]Stanford University [3]Microsoft Research [4]DeepMind, UK. Correspondence to: Ning Miao <ning.miao@stats.ox.ac.uk>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

be usefully applied as a global augmentation, even though it is a useful invariance for the *specific input instance* of a leaf. Similar examples regularly occur for other transformations, such as cropping (see Figure 1b).

Another line of recent work (Zhou et al., 2020; Cheung & Yeung, 2022) has instead looked to utilize instance-aware augmentations by defining a small predefined set of allowable transformations, then introducing a policy that assigns probabilities (and magnitudes) to elements of this set as a function of the input. While these approaches allow some of the shortfalls of global augmentations to be overcome, they do not have the flexibility to learn fine-grained transformation distributions, or uncover underlying invariances.

To address these shortfalls, we introduce InstaAug, a new approach to learn *instance-specific* augmentations by capturing *local* invariances that are specific to the region of the provided input. InstaAug is based on using a transformation distribution of the form $p(\tau;\phi(\mathbf{x}))$, where ϕ is a deep neural network that maps inputs to transformation distribution parameters. We refer to ϕ as an *invariance module*. It can be trained simultaneously with the downstream model in a fully end-to-end manner, or individually with a fixed pre-trained model. Both cases only require access to training data and optimize a single objective function that minimizes the training error while maintaining transformation diversity. As such, *InstaAug* allows one to directly learn powerful and general augmentations, without requiring access to additional data or annotations.

We evaluate InstaAug in both supervised and self-supervised settings, focusing on image classification and contrastive learning respectively. Our experimental results show that InstaAug is able to uncover meaningful invariances that are consistent with human cognition, and improve model performance for various tasks compared with baseline models. While we primarily focus on the case where the invariance module is trained alongside the downstream model (to allow augmentation during training), we find that InstaAug can also provide substantial performance gains when used to learn test-time augmentations for large pre-trained models. Accompanying code is provided at `https://github.com/NingMiao/InstaAug`.

2. Background

Data augmentation methods operate as a wrapper algorithm around some downstream model, f, randomly transforming the inputs $\mathbf{x} \in \mathcal{X}$ before they are passed to the model. The outputs of the augmented model are given by $f(\tau(\mathbf{x}))$, where $\tau : \mathcal{X} \mapsto \mathcal{X}$ represents the transformation, sampled from some transformation distribution $p(\tau)$. The aim of this augmentation is to instill inductive biases into the learned model, leading to improved generalization by capturing invariances of the problem. It can be used both during training to provide additional synthetic training data, and/or at test-time, where ensembling the predictions from multiple transformations can provide a useful regularization that often improves performance (Shanmugam et al., 2021).

Some approaches look to learn aspects of the augmentation (Cubuk et al., 2018; 2020; Lim et al., 2019; Ho et al., 2019; Hataya et al., 2020; Li et al., 2020; Zheng et al., 2022). These approaches can be viewed as learning parameters of $p(\tau)$, helping to automate its construction and tuning. Of particular relevance, Augerino (Benton et al., 2020) provides a mechanism for learning augmentations using a simple end-to-end training scheme, where the parameters of the downstream model and transformation distribution are learned simultaneously using the (empirical) risk minimization

$$\min_{f,\theta} \mathbb{E}_{\mathbf{x},y\sim p_{\text{data}}} \left[\mathbb{E}_{\tau\sim p_\theta(\tau)} \left[\mathcal{L}(f(\tau(\mathbf{x})), y)\right]\right] + \lambda\mathbb{R}(\theta), \quad (1)$$

where $\mathcal{L}$ is a loss function and $\lambda\mathbb{R}(\theta)$ is a regularization term that encourages large transformations.

All of these approaches can be thought of as *global* augmentation schemes, in that transformations are sampled independently to the input. For an unrestricted, universal, class of transformations, this assumption can be justified through the noise outsourcing lemma (Kallenberg & Kallenberg, 1997): any conditional distribution $Y|X = \mathbf{x}$ can be expressed as a deterministic function $g : \mathcal{X} \times \mathbb{R} \to \mathcal{Y}$ of the input and some independent noise $\varepsilon \sim \mathcal{N}(0, I)$. Thus, using reparameterization, the dependency on $\mathbf{x}$ can, in principle, be entirely dealt with by the transformation itself. However, in practice, the transformation class must be restricted to provide the desired inductive biases, meaning this result no longer holds and so the independence assumption can cause severe restrictions. For example, sampling color jitterings independently to the input is equivalent to the unrealistic assumption that the labels of all images $\mathbf{x}$ are invariant to the same group of changes (*cf.* Figure 1a).

3. InstaAug: Capturing Local Invariances

In order to remedy the problems of global augmentations, we propose InstaAug. InstaAug learns an *input dependent* distribution $p(\tau;\phi(\mathbf{x}))$ of information-preserving transformations that actively makes use of the input $\mathbf{x}$ via the *invariance module* ϕ, as opposed to learning a global transformation distribution $p_\theta(\tau)$. This generalizes the hypothesis class of transformation distributions, and significantly increases the flexibility and expressivity of the resulting augmentation, without undermining our ability to carefully control the inductive biases that are imparted. It can also informally be viewed as a mechanism for learning invariances that are local to the specific input.

We argue that a good augmentation strategy needs to fulfill two properties. First, the transformations should preserve

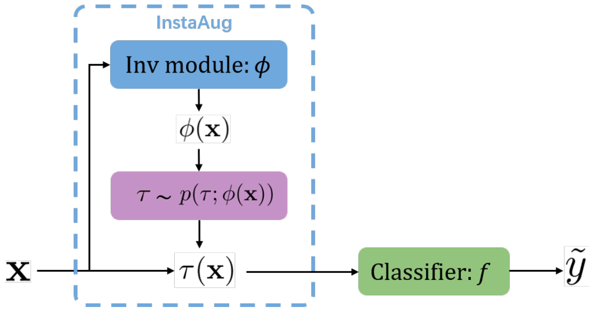
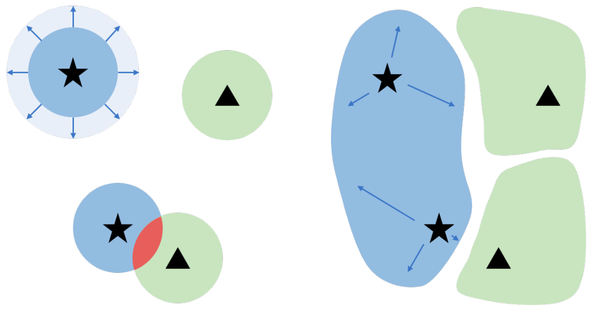


Figure 2: Summary of InstaAug.



(a) Global augmentation (b) InstaAug

Figure 3: InstaAug learns more diverse augmentations that also preserve labels compared to global augmentations. ★ and ▲ are samples from two different classes. Blue and green shades represent label-preserving augmentations for each class. In (a), the upper ★ would benefit from being further augmented, but some of the augmented samples for the lower ★ are already over-augmented and indistinguishable from another class (see the red intersection). InstaAug solves this problem by learning a different augmentation for each instance, as shown in (b).

the information in $\mathbf{x}$ that is necessary for the task at hand. For example, transformations must preserve information about the label for supervised tasks. Second, the set of transformations needs to have sufficient 'diversity' to effectively augment the data; we quantify this as the entropy of the transformation distribution $p(\tau; \phi(\mathbf{x}))$. In addition to their intuitive nature, in Appendix A we provide theoretical analysis that shows these requirements naturally originate from a decomposition of the generalization error between the true risk and augmented empirical risk of f. For simplicity, we describe InstaAug for the specific case where f is a classifier in the remainder of this section.

3.1. Model structure

InstaAug is based around using a simple plug-in invariance module, ϕ, between the input $\mathbf{x}$ and the classifier f, as shown in Figure 2. We assume a parametric family of distributions $p(\tau; \cdot)$ over some transformation space, then use ϕ, which is a trainable neural network, to predict its parameters for a given input. During training, we *sample* a transformation $\tau \sim p(\tau; \phi(\mathbf{x}))$, which is applied to $\mathbf{x}$ to generate an augmented sample $\tau(\mathbf{x})$, before feeding this into the classifier f.

3.2. Training

Good augmentations should induce substantial changes to the input $\mathbf{x}$ while preserving all necessary information about the task at hand, thereby capturing the maximum possible invariance. Figure 3a illustrates the tension between these two objectives experienced by global augmentation schemes. Wider-ranging transformations are generally beneficial for generalization, but 'excessive' transformations will generate samples that will be incorrectly classified. In Figure 3a we see this in the red area, where the augmentations for a pair of data points have started to overlap, creating ambiguity and inevitably misclassifications. Using instance-specific augmentations (Figure 3b) allows for a better trade-off of these needs. However, to achieve this we need our objective to encourage diversity in augmentations, not just low training error. It should also let the level of diversity vary between inputs, as some points will be able to support larger transformations than others.

Based on these needs, training is done by simultaneously minimizing a conventional expected loss with respect to both ϕ and f (or just ϕ if f is a fixed pre-trained classifier as per Section 5.3), while regularizing the average entropy of the transformations, $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\mathbb{H}[p(\tau; \phi(\mathbf{x}))]]$. The core motivation for this setup is that minimizing the expected loss will naturally encourage the information needed for prediction to be preserved, but the regularization on entropy is needed to enforce diversity. Further motivation is provided by the theoretical analysis of Appendix A.

By appropriately parameterizing $p(\tau; \phi(\mathbf{x}))$ (see Section 3.3), we can write down its entropy in closed form. We can then formulate the problem as minimizing the following w.r.t. f and ϕ:

$$\mathbb{E}_{\mathbf{x},y\sim p_{\text{data}},\tau\sim p(\tau;\phi(\mathbf{x}))}\left[\mathcal{L}(f(\tau(\mathbf{x})), y) - \lambda \mathbb{H}[p(\tau; \phi(\mathbf{x}))]\right], \tag{2}$$

where $\mathcal{L}$ is the loss of the downstream task, for which we will generally use the cross-entropy. Unlike in the Augerino objective of Equation (1), λ here is an automatically-tuned weight of the entropy term that enables precise control over transformation diversity. Specifically, we initialize λ with a small positive value, then increase it when the average entropy drops below a lower bound $\mathrm{H}_{\min}$ and decrease it when it exceeds an upper bound $\mathrm{H}_{\max}$) during training. Here $\mathrm{H}_{\min}$ and $\mathrm{H}_{\max}$ are hyperparameters, through which we can

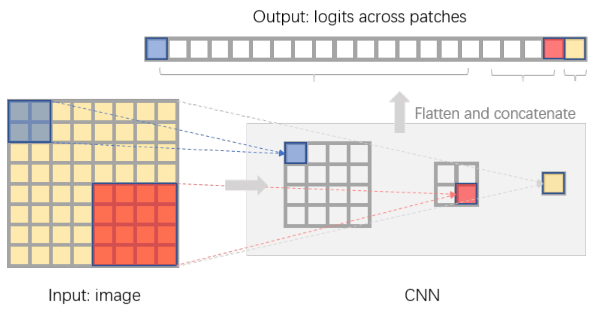


Figure 4: Location-related parameterization of crops by a CNN. The shaded area (bottom right) shows a simplified 3-layer CNN, and squares represent units at different convolutional layers. Each unit defines a patch in the input image (shown in the same color) through its receptive field. The activation value of the unit then gives the corresponding unnormalized log probability for that patch.

directly control the diversity level of learned transformations during the whole training process. As described in Table E.1, they can easily be tuned.

This dynamic λ is necessary because the requirement for λ is different at different stages of training. In the beginning, when the classifier is weak, we need a small λ to avoid the transformations becoming overly diverse, which results in different classes overlapping with each other. As the classifier gets more powerful during training, larger λ is needed to compete with the cross-entropy term $\mathcal{L}$.

Using this approach, the invariance module and downstream model can be trained simultaneously using end-to-end gradient descent, utilizing the reparameterization trick to deal with the stochasticity of τ when possible (Kingma & Welling, 2014), and the REINFORCE estimator (Williams, 1992) otherwise. The approach can also be extended to regression or self-supervised learning by substituting the loss function $\mathcal{L}$ (see Appendix C).

3.3. Parameterization of augmentations

The parameterization method is another critical factor in the quality of learned invariances. A good parameterization should be flexible enough to reflect the complexity of real data while not creating obstacles to gradient-based learning. Here we focus on parameterizing transformations that are frequently used in computer vision, though our framework can easily be extended to other domains. Due to the varied characteristics of different image transformations, we design two different parameterization methods for $p(\tau; \phi(\mathbf{x}))$.

Uniform parameterization. For simpler transformations, such as rotation and color jittering, we find that a uniform distribution is enough for parameterizing $p(\tau; \phi(\mathbf{x}))$, such that $\phi(\mathbf{x})$ returns a pair $(\theta_{\min}, \theta_{\max})$ representing extrema of the possible transformations. For example, for rotations, these represent the maximum and minimum rotation angles, such that $\tau(\mathbf{x}) = R(\theta)\mathbf{x}$, where $\theta \sim \mathcal{U}(\theta_{\min}, \theta_{\max})$ and $R(\theta)$ is the rotation operator. To compose multiple transformations, we simply sample them independently, such that $p(\tau_1, \ldots, \tau_K; \phi(\mathbf{x})) = \prod_{k=1}^{K} p(\tau_k; \phi_k(\mathbf{x}))$. This provides a similar parameterization to (Benton et al., 2020), but where $(\theta_{\min}, \theta_{\max})$ now critically varies with the input $\mathbf{x}$ and there is no symmetry assumption on this range.

Location-related parameterization Using this uniform parameterization is unfortunately not appropriate for more complex transformations like cropping. Firstly, the distribution of crop centers may be multi-modal, since important information may exist in different parts of an image. Secondly, the desired crop size and center are often highly correlated so cannot be sampled independently. Finally, we encountered significant practical training issues when using the uniform parameterization for cropping, with ϕ becoming trapped in local optima with little transformation diversity.

We, therefore, propose an alternative location-related parameterization (LRP) for cropping, which is based on defining a large set of representative crops, then constructing ϕ to map from inputs to a vector of probabilities over this set. As shown in Figure 4, this is achieved using a CNN where each hidden unit corresponds to a possible crop defined by its receptive field. In order to select crops with different sizes, units from different layers are utilized, with those of earlier/latter layers representing smaller/larger crops. This parameterization proved more effective than simply outputting the probabilities from a conventional network, due to the greater parameter sharing between related crops. We note that it can also be directly extended to other transformations, such as masking, local blurring, pixel-wise perturbation, and local color jittering.

4. Related Work

Hard-coded invariance. Many recent works have looked to hard-code global invariance in neural networks. For example, various architectures have been designed to be invariant to translation (Chaman & Dokmanic, 2021; Zhang, 2019), rotation (Worrall et al., 2017; Zhou et al., 2017; Marcos et al., 2017), scaling (Worrall & Welling, 2019; Sosnovik et al., 2019) or other group actions (Cohen & Welling, 2016; Xu et al., 2021). Unfortunately, they require the set of invariant transformations to be closed under composition, leaving out many practical transformations that do not form a group.

Learning augmentations. There have been numerous prior works that automatically learn *global* augmentations and invariance from data. As discussed in Section 2, Augerino (Benton et al., 2020) is perhaps the most closely linked such approach to InstaAug as it also relies on end-to-end training (see Appendix B for further discussion on

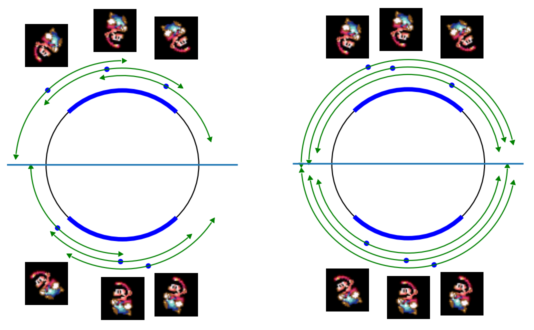


Figure 5: Learned invariances for the Mario and Iggy dataset. The blue arcs show the training data range, while the green arcs show example learned transformation distributions.

its similarities and differences to InstaAug). AutoAugment (Cubuk et al., 2018) instead uses reinforcement learning to find augmentation strategies that increase accuracy on a separate validation set. Various follow-up works have improved its efficiency and/or performance (Lim et al., 2019; Ho et al., 2019; Tang et al., 2019; Hataya et al., 2020; Li et al., 2020; Cubuk et al., 2020; Zheng et al., 2022).

Augmentation policies. A couple of recent works have further looked to learn augmentation *policies* that allow a degree of dependency on the input or class label, namely AdaAug (Cheung & Yeung, 2022) and MetaAugment (Zhou et al., 2021). These policies assign probabilities and magnitudes to a fixed finite list of possible transformation operations. Though they can depend on the input, they only make discrete choices and cannot learn a fine-grained transformation distribution in the way that InstaAug does; they are thus not suitable for capturing local invariances. For example, using InstaAug with cropping learns a joint distribution over patch positions and sizes, whereas these methods only learn a probability for whether to apply cropping or not, and a scalar magnitude to use if it is applied. Further, both AdaAug and MetaAugment require a separate validation set, only consider augmentation during training (so cannot be used for pre-trained classifiers), and cannot be applied in unsupervised settings. AdaAug also has the additional restriction that its policy is based on a linear mapping from the penultimate layer of the classification model, so its input dependence is inherently limited. Meanwhile, although MetaAugment learns a sample-level policy network, in practice it averages this policy among training samples to form a global policy applied to all samples.

Other related work. The spatial transformer (Jaderberg et al., 2015) aims to learn instance-specific transformations, but only applies a single transformation to each input rather than a distribution of transformations, making it distinct from data augmentation. Luo et al. (2020) and Kim et al. (2020a) both also learn instance-specific augmentations. However, the latter consider only test-time augmentation, while the former introduces an approach that is highly specialized to test recognition and cannot be applied in the more general settings we consider. Tamkin et al. (2020) and Chen et al. (2021) both utilize adversarial augmentations to increase robustness. Zhou et al. (2020) learn symmetries shared across several datasets through a meta-learning scheme. Mixup methods (Zhang et al., 2018; Yun et al., 2019; Ramé et al., 2021) can also be thought of as a specific type of data augmentation. Some of them (Kim et al., 2020c;b; Park et al., 2022), allow for input dependence through gradient-based saliency (Simonyan et al., 2013). However, they use a fixed augmentation strategy rather than learning a transformation distribution.

5. Supervised Learning Experiments

5.1. Rotated 2D images

We first consider a simple synthetic dataset proposed in Benton et al. (2020). The dataset contains four categories, (1) upright Mario; (2) upside-down Mario; (3) upright Iggy; and (4) upside-down Iggy. Each of the four base images is randomly rotated in the interval of $[-\pi/4, \pi/4]$ to form the training dataset. The task is to predict the correct character (Mario vs Iggy) and the orientation (up vs down). We assess whether InstaAug is able to learn the 'best' rotation range for each sample—i.e. the maximum range that avoids 'up' and 'down' classes from overlapping.

Figure 5 shows that InstaAug effectively recovers the broadest range of rotations for each image while preserving labels, while Augerino only learns a subset of these ranges. This can be most easily seen by the fact that the transformation distributions (shown in green) always extend to very close to the true class boundary for InstaAug, but not for Augerino. These gains are because Augerino learns a *single* global augmentation distribution shared across all images (note the shared transformation distribution arcs), which are inevitably limited for any given input.

5.2. Cropping

We now move to more realistic images and to the most common and effective form of image augmentation: cropping. We first evaluate the performance of jointly training InstaAug and the classifier on Tiny-Imagenet (TinyIN, 64×64), as it inherits the image complexity of ImageNet whilst being within our computational budget. TinyIN is a standard testbed for data augmentations. Full experiment details are given in Appendix D.1.

We benchmark InstaAug alongside several augmentation baselines, including random crop, Augerino, and AdaAug. For random crop, we use a uniform distribution on patch

Table 1: Test accuracy for Tiny-ImageNet with cropping augmentation. The Instance column indicates whether the method is instance-specific or not. Statistics are computed over 10 runs, except for MixUp methods, whose results are directly taken from their respective papers. We omit comparison to other global augmentation schemes, as these only learn the size ranges of the cropping, which is already covered by the hyperparameter tuning of Random crop.

Method	Instance	Accuracy (%)
No augmentation	✗	55.06$\pm$0.10
Random crop	✗	64.49$\pm$0.12
MixUp	✗	63.74
CutMix	✗	65.09
MixMo	✗	64.80
Puzzle-Mix	✓	63.48
Co-Mixup	✓	64.15
Saliency grafting	✓	64.84
Augerino	✗	55.02$\pm$0.29
AdaAug	✓	64.03$\pm$0.19
-w/ LRP	✓	62.01$\pm$0.23
InstaAug (ours)	✓	**66.02**$\pm$0.18
-w/o LRP	✓	55.39$\pm$0.19
-w/o input	✗	63.20$\pm$0.12
-class-specific	✗/✓	60.55$\pm$0.50

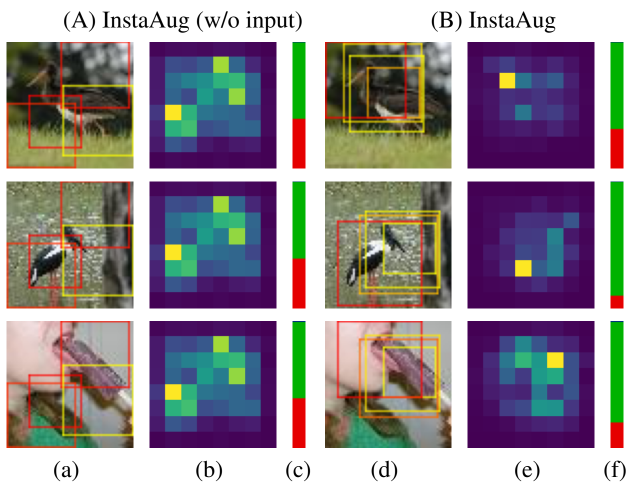


Figure 6: InstaAug (B) learns more sensible crops compared to random and learned global (A) augmentations. Columns (a, d) show examples of sampled crops, with red edges indicating higher probability. Columns (b, e) show density maps for the crop centers, with brighter color meaning higher probability. Columns (c, f) give the proportion of crops (red) above a particular size threshold, showing that InstaAug produces fewer large crops.

size and position, tuning the bounds on the former through a comprehensive hyperparameter sweep to ensure appropriate scaling. We further compare to other prior works that have obtained competitive results on TinyIN (Zhang et al., 2018; Yun et al., 2019; Ramé et al., 2021; Kim et al., 2020c;b; Park et al., 2022), directly taking their reported results.

In order to ablate the effects of input-dependency and our location-related parameterization (LRP, see Figure 4) on InstaAug, we additionally assess the performance of *InstaAug (without LRP)* by using same uniform parameterization as Augerino; *InstaAug (without input)* that uses the LRP and general InstaAug setup, but shares the transformation distribution across all inputs rather than learning an input-specific augmentation; and *InstaAug (class-specific)*, which takes training labels instead of images as inputs. Test-time augmentation using 50 transformation samples is deployed for all variants of InstaAug, along with the Augerino, AdaAug, and random augmentation baselines (see Appendix C.2). For InstaAug (class-specific), this test-time augmentation is based on random cropping, as class information is not available at test-time and simply omitting test-time augmentation performed poorly. Following prior works, we choose the PreActResNet-18 architecture (He et al., 2016b) with width $= 1$ as the classifier for all methods.

Table 1 shows the (top-1) test accuracy for each method. In agreement with prior works, we find that random cropping increases accuracy by 9.4% over no augmentation, which is achieved where cropping scale $= [0.1, 1]$. InstaAug outperforms random cropping and its own global version without input by 1.5% and 2.8% respectively, highlighting the effect of learning instance-specific augmentation.

Allowing only for class dependence actually produces even worse performance than just ignoring the input completely, presumably because of the inevitable resulting mismatch in the augmentations used in training and testing. Methods with mean-field uniform parameterization (including Augerino and InstaAug without LRP) performed extremely poorly, noticeably worse than just random cropping. This is because they were found to become easily stuck at local minima with low cropping diversity, leading to similar performance as *no augmentation*. The original version of AdaAug achieves similar performance to Random crop, but is incapable of dealing with the large search space of LRP, leading to a small reduction in performance when this is added. Note that the potentially unexpectedly good performance of the random cropping baseline compared to the other global baselines stems from the careful hyperparameter sweep used to tune its crop size, which proved more effective than these more direct training mechanisms. See Appendix E.3 for more discussion.

Figure 6 shows example crops and learned transformation distributions for InstaAug and a global augmentation scheme (InstaAug without input). We see that InstaAug is able to learn a cropping scheme that focuses on the key aspect of the input image, while the baselines cannot.

Table 2: InstaAug boosts the test accuracy (%) with test-time augmentation on Imagenet. Invariance modules learned on ResNet-50 can also be directly applied to other models such as ResNet-18 and XCiT to improve generalization without fine-tuning. By contrast, global augmentation schemes are actually detrimental to test-time augmentation.

Method	#Sample	ResNet50	ResNet18	XCiT
No aug	1	80.43	69.73	86.34
Random crop	4	78.45±0.04	66.13±0.04	82.05±0.01
AutoAug	4	77.84±0.05	59.50±0.01	81.40±0.00
FastAutoAug	4	77.87±0.06	61.43±0.02	81.42±0.01
InstaAug	4	**80.92**±0.04	**70.59**±0.05	**86.43**±0.04
Random crop	10	79.60±0.01	67.87±0.01	82.84±0.00
AutoAug	10	79.20±0.04	63.96±0.03	82.43±0.02
FastAutoAug	10	79.28±0.01	65.65±0.02	82.45±0.02
InstaAug	10	**81.18**±0.02	**70.96**±0.03	**86.47**±0.02

5.3. Applying InstaAug to a fixed classifier

InstaAug can also be used to learn suitable augmentations for a fixed pre-trained classifier. This can most notably be useful as a means to learn test-time augmentations. As the invariance module is itself only a small network, it can be done relatively cheaply, even when the dataset and downstream model are very large. We exploit this on the larger Imagenet dataset (224×224) (Deng et al., 2009), again focusing on cropping augmentations and utilizing the LRP parameterization from Section 3.3.

Training the invariance module in this setting is done in exactly the same way as elsewhere, using the training procedure of Section 3.2 with the normal training data. The only thing that is changed is that f is now fixed to a pre-trained classifier—specifically, the ResNet-50 (He et al., 2016a) from Wightman (2019) (which did not use an invariance module during training)—rather than being simultaneously learned. We are thus simply learning invariances, without affecting the training of f.

In Table 2, we show the effect of using the learned invariance module for test-time augmentation, finding that it is able to noticeably improve accuracy, unlike the baseline test-time augmentations of random cropping, AutoAugment (Cubuk et al., 2018), and Fast AutoAugment (Lim et al., 2019). Note that AdaAug cannot be used in this fixed-classifier setting.

In order to evaluate the generalization performance of our learned augmentation module, we further apply the augmentation trained on ResNet-50 to two different models *with zero fine-tuning*: ResNet-18 (He et al., 2016a) and XCiT (Ali et al., 2021). We find that the learned augmentation transfers very effectively to these different models, which implies that the local invariances InstaAug learns to reflect the natural invariances of the underlying classification problem, rather than being specific to the model that was used to train the augmentation module.

Table 3: InstaAug achieves higher general accuracy than baseline methods when trained on D45 (Daylight, 4500K).

Method	Test aug?	Accuracy (%)
No aug	✗	$72.87_{\pm 0.10}$
Random aug	✗	$79.99_{\pm 0.13}$
Augerino	✗	$78.97_{\pm 0.10}$
AdaAug	✗	$75.27_{\pm 0.30}$
InstaAug	✗	**81.11**$_{\pm 0.20}$
Random aug	✓	$80.55_{\pm 0.16}$
Augerino	✓	$79.34_{\pm 0.14}$
AdaAug	✓	$76.43_{\pm 0.15}$
InstaAug	✓	**81.35**$_{\pm 0.19}$

Table 4: InstaAug significantly outperforms baseline methods in general test accuracy (%) on different difficulty levels. Difficulty level is controlled by the number of randomly sampled lighting conditions seen. Test-time augmentation is included for random and InstaAug and we repeat each experiment for 10 times.

#Lighting conditions	1	2	4	8
No aug	$68.5_{\pm 2.6}$	$78.1_{\pm 1.8}$	$84.8_{\pm 0.7}$	$87.8_{\pm 0.5}$
Random augmentation	$72.7_{\pm 2.7}$	$80.8_{\pm 1.3}$	$85.9_{\pm 0.6}$	$87.3_{\pm 0.3}$
InstaAug	**76.0**$_{\pm 2.5}$	**83.6**$_{\pm 1.1}$	**88.2**$_{\pm 0.5}$	**89.6**$_{\pm 0.3}$

5.4. Color jittering on textures

Color jittering is another important type of data augmentation, which can help models generalize to different lighting conditions. We benchmark on the texture classification dataset RawFooT (Bianco et al., 2017). RawFooT includes 68 different samples of raw food and each sample has an image taken under each of 46 lighting conditions (see Figure C.1 for examples), which makes it an ideal testbed to investigate methods' generalization ability between different lighting conditions. We crop the original images to create the train set and test set. For each original image with a resolution of 800×800, we randomly sample 200 different 200×200 patches in the upper half as training images. The same procedure is taken on the lower half to produce test images, giving a train set and a test set for each different lighting condition. To evaluate the generalization ability to a broader range of lighting conditions, we evenly mix test images from all lighting conditions to form a general test set, while controlling the conditions during training.

We first train on a single lighting condition D45 (4500K, daylight) resembling natural light. Table 3 shows that InstaAug outperforms all baselines with and without test-time augmentation. We find that Augerino (with relaxed symmetry restrictions on learned intervals) underperforms random augmentation because its parameters ϕ are often stuck near their initial values. We believe this is due to the conservative nature of using global augmentations (*cf.* Figure 3), where even a small change in the parameters may largely increase

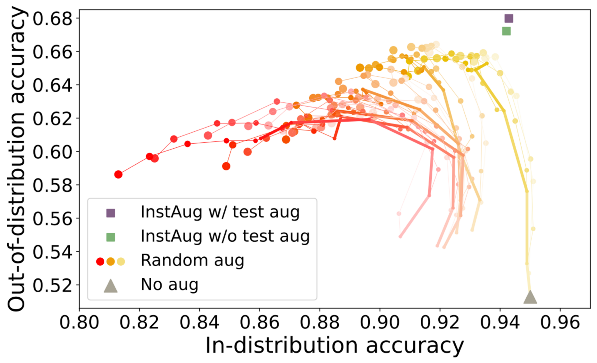


Figure 7: In- and out-of-distribution test accuracy for models trained on RawFooT D45. The round dots are random augmentation with different hyperparameter settings. The colors of dots change from yellow to red as hue jittering increases; more saturated dots indicate higher saturation jittering; larger dots mean higher brightness jittering. Thick lines connect dots with the same hue and brightness jitter, thin lines link dots with the same hue and saturation jitter.

the training loss, which prohibits wide-ranging augmentations. AdaAug does not perform well either, which might be a result of its inability to learn the interval length for the distribution of each transformation.

We also compare in-distribution and out-of-distribution generalization by splitting the 46 test sets into two groups, according to the similarity of their lighting conditions to D45—see Appendix D.2 for the details on the splitting method. In Figure 7 we can see that above a certain in-distribution performance, there exists a trade-off for random augmentation between in-distribution accuracy and out-of-distribution generalization, controlled through the hyperparameter settings. InstaAug, meanwhile, delivers higher out-of-distribution performance than any of the hyperparameter configurations, while also simultaneously giving better in-distribution accuracy to the vast majority of them as well.

We can further vary the difficulty of the classification task by using different numbers of lighting conditions in the training data. In Table 4, we randomly select a set number of lighting conditions to use as the training set for each baseline. As expected, the accuracy increases with the number of lighting conditions for all methods. However, the effect of random augmentation saturates: it performs similarly to no augmentation with 8 lighting conditions. By contrast, InstaAug always provides improvements. In Appendix D, we show that these gains come at very little computational overhead at both train and test time.

6. InstaAug for Contrastive Learning

Contrastive learning aims to learn features that are approximately invariant to certain augmentations. Typical contrastive learning methods, such as SimCLR (Chen et al., 2020; Ermolov et al., 2021), first sample two independent transformations, $\tau_1, \tau_2 \sim p(\tau)$, and apply them to an input image $\mathbf{x}$, generating two views $\mathbf{x}_1$ and $\mathbf{x}_2$. They then feed the transformed images to a neural encoder f, which is trained to maximize the similarity between $f(\mathbf{x}_1)$ and $f(\mathbf{x}_2)$, measured with a contrastive loss.

Table 5: Representations learned by InstaAug perform better in the downstream linear classification task than baselines. *Results of Un-Mix are directly taken from (Shen et al., 2022), which has the same network structure (ResNet-18), training algorithms (SimCLR) and linear classifier as ours.

Method	Accuracy (%)
Un-Mix (Shen et al., 2022)	49.58^*
Random crop	$51.63_{\pm 0.30}$
InstaAug (without input)	$54.20_{\pm 0.23}$
InstaAug	$\mathbf{55.05}_{\pm 0.21}$

The choice of augmentations directly influences the learned invariance of the encoder and thus forms a crucial ingredient of contrastive learning (Bachman et al., 2019; Chen et al., 2020; Tian et al., 2020). Existing schemes use global augmentations that often introduce unrealistic assumptions. For example, if there are multiple entities in an image, such as grass and cattle in Figure 1b, random cropping will pull features for different entities closer to each other. Consequently, we propose InstaAug as a more flexible instance-specific augmentation method for contrastive learning.

Applying InstaAug to contrastive learning is similar to the supervised case shown in Section 3. The main difference is, given an input $\mathbf{x}$, we sample two τ independently from the input-specific distribution $p(\tau; \phi(\mathbf{x}))$, before they are applied to $\mathbf{x}$. The training objective is correspondingly changed to minimizing the contrastive loss while keeping the diversity in a reasonable range.

We again consider TinyIN and evaluate three methods: InstaAug, InstaAug (without input), and Random crop. We exclude methods with uniform parameterization because of their earlier poor performance and note that AdaAug is not applicable for unsupervised learning. All experiments are based on the SimCLR framework and use the PreActResNet-18 network as the encoder. We train each model with a batch size of 512 for 500 epochs. We then train a linear classifier to evaluate feature quality. We use test-time augmentation—with 10 sampled crops—as this has been shown to improve performance (Foster et al., 2021).

Table 5 shows that InstaAug outperforms the random and global augmentation schemes as well as Un-Mix (Shen et al., 2022), which is a recent variant of MixUp methods for contrastive learning. We see from the examples in Figure 8 that InstaAug focuses on the salient features containing

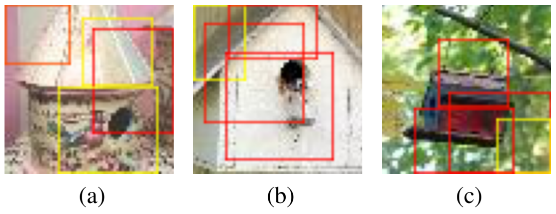


Figure 8: Examples of learned croppings by InstaAug for contrastive learning. Conventions as per Figure 6.

important information. We also notice that the sizes of learned patches are correlated to the sizes of the objects in the images. Thus, InstaAug is able to learn sensible instance-specific augmentations in a fully unsupervised setting.

7. Conclusions

In this paper, we introduced InstaAug, a method for learning instance-specific data augmentations that capture local invariances of the underlying data-generating process. This is achieved by training an augmentation module that parametrizes an input-dependent distribution over transformations, whose samples can be used to augment the training data on the fly and/or for test-time augmentation. The main benefits of InstaAug stem from its applicability to a wide range of settings, its ease of use, and crucially its capacity to learn meaningful augmentations that in turn improve performance. Empirically, we have demonstrated these benefits for both classification and contrastive learning problems, considering several classes of transformations.

Acknowledgements

We would love to thank Michael Hutchinson, Bryn Elesedy, and Sheheryar Zaidi for valuable discussions. We would also like to thank Google Cloud Platform (GCP) for their generous help in computing resources. Ning Miao gratefully acknowledges funding from China Scholarship Council - University of Oxford Scholarship as well as Tencent AI Labs through the Oxford-Tencent Collaboration on Large Scale Machine Learning.

References

Ali, A., Touvron, H., Caron, M., Bojanowski, P., Douze, M., Joulin, A., Laptev, I., Neverova, N., Synnaeve, G., Verbeek, J., et al. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems*, 34: 20014–20027, 2021.

Bachman, P., Hjelm, R. D., and Buchwalter, W. Learning representations by maximizing mutual information across views. *Advances in neural information processing systems*, 32, 2019.

Benton, G., Finzi, M., Izmailov, P., and Wilson, A. G. Learning invariances in neural networks. *Advances in Neural Information Processing Systems*, 2020.

Bianco, S., Cusano, C., Napoletano, P., and Schettini, R. Improving cnn-based texture classification by color balancing. *Journal of Imaging*, 3(3):33, 2017.

Chaman, A. and Dokmanic, I. Truly shift-invariant convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3773–3783, 2021.

Chen, S., Dobriban, E., and Lee, J. H. Invariance reduces variance: Understanding data augmentation in deep learning and beyond. *CoRR*, abs/1907.10905, 2019. URL http://arxiv.org/abs/1907.10905.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Chen, X., Xie, C., Tan, M., Zhang, L., Hsieh, C.-J., and Gong, B. Robust and accurate object detection via adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16622–16631, 2021.

Cheung, T.-H. and Yeung, D.-Y. Adaaug: Learning class- and instance-adaptive data augmentation policies. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=rWXfFogxRJN.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning*. PMLR, 2016.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. *Conference on Computer Vision and Pattern Recognition*, 2018.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Ermolov, A., Siarohin, A., Sangineto, E., and Sebe, N. Whitening for self-supervised representation learning. In *International Conference on Machine Learning*, pp. 3015–3024. PMLR, 2021.

Foster, A., Pukdee, R., and Rainforth, T. Improving transformation invariance in contrastive representation learning. In *International Conference on Learning Representations*, 2021.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

Hataya, R., Zdenek, J., Yoshizoe, K., and Nakayama, H. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision*, pp. 1–16. Springer, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Ho, D., Liang, E., Chen, X., Stoica, I., and Abbeel, P. Population based augmentation: Efficient learning of augmentation policy schedules. In *International Conference on Machine Learning*, pp. 2731–2741. PMLR, 2019.

Jaderberg, M., Simonyan, K., Zisserman, A., et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

Kallenberg, O. and Kallenberg, O. *Foundations of modern probability*, volume 2. Springer, 1997.

Kim, I., Kim, Y., and Kim, S. Learning loss for test-time augmentation. *Advances in Neural Information Processing Systems*, 33:4163–4174, 2020a.

Kim, J., Choo, W., Jeong, H., and Song, H. O. Co-mixup: Saliency guided joint mixup with supermodular diversity. In *International Conference on Learning Representations*, 2020b.

Kim, J.-H., Choo, W., and Song, H. O. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *International Conference on Machine Learning*, pp. 5275–5285. PMLR, 2020c.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Li, Y., Hu, G., Wang, Y., Hospedales, T., Robertson, N. M., and Yang, Y. Dada: differentiable automatic data augmentation. *arXiv preprint arXiv:2003.03780*, 2020.

Lim, S., Kim, I., Kim, T., Kim, C., and Kim, S. Fast autoaugment. *Advances in Neural Information Processing Systems*, 2019.

Luo, C., Zhu, Y., Jin, L., and Wang, Y. Learn to augment: Joint data augmentation and network optimization for text recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13746–13755, 2020.

Lyle, C., van der Wilk, M., Kwiatkowska, M., Gal, Y., and Bloem-Reddy, B. On the benefits of invariance in neural networks. *arXiv preprint arXiv:2005.00178*, 2020.

Marcos, D., Volpi, M., Komodakis, N., and Tuia, D. Rotation equivariant vector field networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

Mikołajczyk, A. and Grochowski, M. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, 2018.

Park, J., Yang, J. Y., Shin, J., Hwang, S. J., and Yang, E. Saliency grafting: Innocuous attribution-guided mixup with calibrated label mixing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7957–7965, 2022.

Perez, L. and Wang, J. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

Ramé, A., Sun, R., and Cord, M. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 823–833, 2021.

Shanmugam, D., Blalock, D., Balakrishnan, G., and Guttag, J. Better aggregation in test-time augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1214–1223, 2021.

Shen, Z., Liu, Z., Liu, Z., Savvides, M., Darrell, T., and Xing, E. Un-mix: Rethinking image mixtures for unsupervised visual representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 2216–2224, 2022.

Shorten, C. and Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Sosnovik, I., Szmaja, M., and Smeulders, A. Scale-equivariant steerable networks. In *International Conference on Learning Representations*, 2019.

Tamkin, A., Wu, M., and Goodman, N. Viewmaker networks: Learning views for unsupervised representation learning. In *International Conference on Learning Representations*, 2020.

Tang, Z., Peng, X., Li, T., Zhu, Y., and Metaxas, D. N. Adatransform: Adaptive data transformation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2998–3006, 2019.

Tian, Y., Sun, C., Poole, B., Krishnan, D., Schmid, C., and Isola, P. What makes for good views for contrastive learning? *Advances in Neural Information Processing Systems*, 33:6827–6839, 2020.

Wightman, R. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Worrall, D. and Welling, M. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems*, 2019.

Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Xu, J., Kim, H., Rainforth, T., and Teh, Y. Group equivariant subsampling. *Advances in Neural Information Processing Systems*, 2021.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

Zhang, R. Making convolutional networks shift-invariant again. In *International conference on machine learning*. PMLR, 2019.

Zheng, Y., Zhang, Z., Yan, S., and Zhang, M. Deep autoaugment. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=St-53J9ZARf.

Zhou, A., Knowles, T., and Finn, C. Meta-learning symmetries by reparameterization. *arXiv preprint arXiv:2007.02933*, 2020.

Zhou, F., Li, J., Xie, C., Chen, F., Hong, L., Sun, R., and Li, Z. Metaaugment: Sample-aware data augmentation policy learning. *Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021.

Zhou, Y., Ye, Q., Qiu, Q., and Jiao, J. Oriented response networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

A. Theoretical Analysis of Generalization Error

We now provide a decomposition of the generalization error—i.e. the difference between the true risk and the training risk—when using ϕ during training of the downstream classifier f. Here we can view the objective of augmentation as adjusting the training objective to encourage the learned model to have a low true risk. As such, the generalization error provides a measure of the effectiveness of the augmentation for the training of f; by analyzing the behavior of the generalization error as a function of the augmentation module, we can derive a characterization of the desirable properties of the latter.

To start our analysis, we first define the true risk of the downstream model, f, as

$$R(f) := \mathbb{E}[\mathcal{L}(f(X), Y)] \tag{A.1}$$

where $(X, Y) \sim p_{\text{true}}(X, Y)$ are drawn from the true data generating distribution. In practice, one might also perform test-time augmentation, implying a different predictive function and thus different true risk, but for the purposes of our analysis, we will assume that this is not done, as this allows us to focus on the impact the invariance module has on f during training.

On the other hand, the implied training risk (i.e. our objective for training f) when using an invariance module is the augmented empirical risk

$$\hat{R}(f, \phi) := \mathbb{E}[\mathcal{L}(f(\tau(x_i)), y_i)] \tag{A.2}$$

where $i \sim \text{Uniform}\{1, \dots, N\}$ is a uniformly sampled index for a point in the original training dataset $\{x_n, y_n\}_{n=1}^{N}$ and $\tau | i \sim p(\tau; \phi(x_i))$ is the sampled transformation. Note that the expectation in Equation (A.2) is only over i and τ, with the data-points themselves not considered random variables for our purposes, because we are only provided with a single fixed training dataset.

The generalization error can now be defined as $\hat{R}(f, \phi) - R(f)$. At a high level, we are interested in finding a ϕ that ensures this has a low magnitude. More precisely, we want ϕ to ensure that the minimizer of the training risk, $\hat{f}^* := \arg\min_f \hat{R}(f, \phi)$, gives as low a true risk, $R(\hat{f}^*)$, as possible. Therefore, we want to keep the generalization error magnitude small across different f (relative to the corresponding variations in $\hat{R}(f, \phi)$ itself), so that the optima of the training and true risks are as similar as possible. In other words, we want a ϕ that ensures $\hat{R}(f, \phi) - R(f)$ is small (in magnitude) for *all* f, especially those close to $\hat{f}^*$. If we do hypothetically drive the generalization error to zero for all f, we will have a mechanism for directly training to the true risk using a finite original training dataset.

To aid with decomposing the generalization error, it is convenient to further define the following random variables through their conditional distributions:

$$\hat{Y} | i \sim p_{\text{true}}(Y = \hat{Y} | X = x_i) \quad \text{with} \quad \hat{Y} \perp\!\!\!\perp \tau, \tag{A.3}$$
$$\tilde{Y} | i, \tau \sim p_{\text{true}}(Y = \tilde{Y} | X = \tau(x_i)). \tag{A.4}$$

We can now write down our decomposition as follows:

$$\begin{aligned}\hat{R}(f, \phi) - R(f) = &\underbrace{\mathbb{E}[\mathcal{L}(f(\tau(x_i)), \hat{Y}) - \mathcal{L}(f(\tau(x_i)), \tilde{Y})]}_{\text{(A)}} \\ &+ \underbrace{\mathbb{E}[\mathcal{L}(f(\tau(x_i)), \tilde{Y}) - \mathcal{L}(f(X), Y)]}_{\text{(B)}} \\ &+ \underbrace{\mathbb{E}[\mathcal{L}(f(\tau(x_i)), y_i) - \mathcal{L}(f(\tau(x_i)), \hat{Y})]}_{\text{(C)}}.\end{aligned} \tag{A.5}$$

From this, we see that if the magnitude of (A), (B), and (C) are all small, then our generalization error magnitude will be small as well. Moreover, if we can construct a ϕ such that these terms are small for *all* f, then we can ensure effective generalization performance. We will now look at each term individually.

(A) provides a precise characterization of how well our transformation preserves the label distribution; it is the difference between the expected loss under the true label distribution of the untransformed inputs and the expected loss under the true label distribution of the transformed inputs, making predictions using the transformed inputs in both cases. In particular, by noting that we have

$$\text{(A)} = \mathbb{E}\left[\mathbb{E}\left[\mathcal{L}(f(\tau(x_i)), \hat{Y}) - \mathcal{L}(f(\tau(x_i)), \tilde{Y}) \middle| i, \tau\right]\right] \tag{A.6}$$

where $f(\tau(x_i))$ is deterministic given τ and i, we have that $\tilde{Y}|i, \tau, \overset{\text{d}}{=} \hat{Y}|i, \forall i, \tau$ is a sufficient (but not necessary) condition to ensure $(\text{A}) = 0$ for all f.[1] That is, it is zero for all f if the conditional distribution on the labels is the same for both the original and transformed inputs for all possible pairs (i, τ), i.e. all possible original inputs and sampled transformations. One simple way to ensure this is to have τ always be equal to the identity mapping, so this term prefers limited transformations.

By contrast, if the transformation destroys information about the label, $\hat{Y}|i$ and $\tilde{Y}|i, \tau$ will now differ, such that, in general, (A) $\neq 0$ and, moreover, it will vary with f. Here we typically expect that (A) ≥ 0,[2] as we are making predictions using the transformed inputs, so the expected loss

[1]Note that $\hat{Y} \overset{\text{d}}{=} \tilde{Y}$ alone is not generally sufficient, as matching in marginal distribution does not ensure that the joint distributions with i and τ also match, in turn yielding different expectations.

[2]Note, though, that this is not formally guaranteed, even for the

under the true label distribution for the transformed inputs will tend to be less than that when labels are generated using the untransformed input. To keep the magnitude of (A) low, we need to ensure that transformations maintain the conditional label distribution as well as possible, i.e. that transformations preserve all input information that is salient for predicting labels.

Conveniently, minimizing $\hat{R}(f, \phi)$ with respect to ϕ, as done by the InstaAug training setup of Section 3.2, will naturally try to reduce (A). Given we expect the term to typically be positive, this provides an explanation for why InstaAug can be effective without any separate consideration in the objective for the need for transformations to maintain the class label distribution.

(B) represents how well our transformation captures the true input distribution. Here we can utilize the fact that, by the definition of $\tilde{Y}$,

$$\mathbb{E}\left[\mathcal{L}(f(\tau(x_i)), \tilde{Y})\middle|\tau(x_i) = x\right] = \mathbb{E}\left[\mathcal{L}(f(X), Y)|X = x\right] \\ =: r(x) \tag{A.7}$$

to write it as

$$\text{(B)} = \mathbb{E}[r(\tau(x_i))] - \mathbb{E}[r(X)], \tag{A.8}$$

where $r : \mathcal{X} \mapsto \mathbb{R}^+$ maps inputs to their true expected loss. We thus see that $\tau(x_i) \overset{\text{d}}{=} X$ is a sufficient (but not necessary) condition to ensure that (B) $= 0$ for all f. That is (B) is always 0 if the process of choosing one of the training inputs at random followed by applying a sampled transformation to that input produces samples distributed exactly according to the true input distribution. Unlike for (A), there is no simple scenario in which we can ensure this is true, with the use of the identity transformation now likely to give significant discrepancies by failing to provide sufficient coverage of the input space: though the x_i may originally have been sampled from $p_{\text{true}}(X)$, there is only a finite set of them, such that repeated sampling from this finite set represents a substantially different distribution to $p_{\text{true}}(X)$. In fact, (B) nicely encapsulates the desire to perform augmentation in the first place, by showing how it can be used to increase the coverage of the input space.

How to best manage Term (B) will vary depending on the type of model used and the form of our transformations. In some situations, it may be that no matter how diverse our transformations are within the class of those allowable, $\tau(x_i)$ will still only cover a subset of the support of X. Here the most important factor for keeping (B) small will be to maximize the diversity of the transformations, e.g. by maximizing their entropy, to ensure the best possible coverage of the true input space. In other cases, it might also be possible to "over–diversify" the inputs, such that $\tau(x_i)$ can become more diffuse than X for some choices of ϕ, potentially causing training to lack focus on the particular test-time input distribution we care about. Here we may need to ensure that the entropy of the transformation does not become so large as to cause such over-diversification, creating a more complex trade-off with the need to ensure sufficient coverage. These two scenarios respectively motivate the lower and upper bounds on the transformation distribution entropy used when training the augmentation module.[3]

For augmentation of high-dimensional data, the former, coverage-limited, scenario is expected to be significantly more likely, as our original training data will generally provide quite poor coverage of the true input distribution, while our transformations will not generally be sufficiently powerful to produce unrepresentative inputs. Moreover, when working with large deep learning models, prediction in one region of the input space is rarely harmed by the addition of data in another input region. Thus, for the typical scenarios, we expect InstaAug to be deployed in, increasing the entropy of the transformations will directly relate to reducing the magnitude of (B). Note here that it will typically be the case that (B) < 0 provided that the transformations maintain the label distribution, as the accuracy of the downstream model will typically be higher for the transformations of the original training data that for the test data.

Term (C) is the error from the fact that we only have one sample of the label for each original training input, rather than the full label distribution. As $\hat{Y} \perp\!\!\!\perp \tau$, we have limited ability to reduce it through controlling ϕ; it essentially represents the irreducible noise in $\hat{R}(f, \phi)$ from only having a finite number of true labels. Note that it is not related to the model's ability to generalize to unseen inputs, as it is based on variability in other possible labels we might have seen for our training inputs themselves; if $Y|X$ is actually deterministic, it is exactly zero. As such, it is of limited interest for our analysis, while it will thankfully generally be much smaller than the other terms for practical problems unless we have both a very small dataset and a very noisy true label distribution.

Putting everything together, we see that (A) and (B) respectively encapsulate the competing needs of the invariance module to maintain the conditional label distribution (i.e. preserve the label information) and maximize cover-

cross entropy loss and an f that exactly captures the true distribution. This is because, while Gibbs' inequality ensures the optimal q given p for a cross-validation expected loss $\mathbb{E}_{p(Y)}[-\log q(Y)]$ is $q = p$, in general, the optimal p given q is not $p = q$.

[3]Note here that the entropy bounds in Section 3.2 are on are on the entropy on the parameters of τ, rather than $\tau(x_i)$ itself. This is because it is difficult to directly control the latter during the training, with the former providing a more practical proxy that is expected to generally be representative.

age of the input space. We have also seen that the former is typically naturally taken care of by minimizing $\hat{R}(f, \phi)$ with respect to ϕ, motivating the cross-entropy term in Equation (2), but the latter requires separate consideration, which we deal with through the regularization term.

B. Details of Augerino

As a method to learn invariance, Augerino (Benton et al., 2020) is quite different from the previous approaches, which usually require an extra validation set. The basic idea behind Augerino is to use a few parameters (θ) to control the transformation distribution on input images and learn these parameters with the training loss of the classifier. Specifically, it minimizes the loss

$$\mathcal{L}_\lambda(\mathbf{x}; y) \triangleq \mathbb{E}[\mathcal{L}(f(\tau(\mathbf{x})); y)] + \lambda \cdot \mathbb{R}(\theta), \qquad \text{(B.1a)}$$

where $\mathcal{L}(\mathbf{x}; y)$ is the cross-entropy loss and $\mathbb{R}(\theta)$ is a regularization function on the volume of the support of the distribution weighted by the hyper-parameter λ.

Comparison with InstaAug. InstaAug shares with Augerino the ideas of tuning augmentation parameters by the classifier loss and using test time augmentation to boost performance, but they are different in the following aspects. The most significant difference is that InstaAug is instance-specific, while Augerino learns global augmentations. Besides, Augerino uses a single scalar θ to parameterize a symmetric uniform distribution ($\mathcal{U}[-\theta, \theta]$) over each type of transformations, which lacks the flexibility to model more complex augmentations, such as cropping.

In addition, Augerino uses a fixed weight λ to balance the training loss and augmentation diversity. However, we find that, in more complicated settings, this is quite impractical. Specifically, we need different λ in different stages of training. If we use a large λ from the start of training, the diversity will quickly diverge to maximum, because the classifier is very weak and the loss is consequently dominated by the diversity term. This will block the training of the classifier because transformed samples from different classes are quite mixed with each other. Otherwise, if we choose a small λ, the diversity will converge to zero after a few epochs, yielding similar results as the vanilla model without augmentation. In neither of the case can we learn a useful augmentation. Consequently, InstaAug directly constrains the diversity to keep it stable during training.

C. Method details

C.1. Regression and self-supervised learning

In Section 3, we use classification as an example to introduce InstaAug. However, InstaAug can be easily applied to other tasks including regression and self-supervised learning. For regression, the classifier (see Figure 2) is replaced by a regressor and the loss function $\mathcal{L}$ in Equation (2) is changed accordingly to absolute or square error. For self-supervised contrastive learning, we replace the classifier and cross-entropy loss with the feature extractor and contrastive loss (such as SimCLR loss (Chen et al., 2020)), respectively. In addition, the sampler samples 2 rather than 1 transformations to generate multiple views for an input $\mathbf{x}$.

Algorithm 1 Location related parameterization

```
Input: Image $\mathbf{x}$, layer number $n\_layer$, channel number $M_i$
Output: Probability of patches $\mathbf{P}_{\text{crop}}$
$\mathbf{F}'_0 = \mathbf{x}$
for ( $i = 1$; $i \leq n\_layer$; $i = i + 1$ )
    $\mathbf{F_i}$ = Conv2d($\mathbf{F}'_{\mathbf{i-1}}$, kernel=2, stride=1, output_channel=$M_i$)
    $\mathbf{F}'_{\mathbf{i}}$ = Pooling($\mathbf{F_i}$, kernel=2) ;   // Conv and pooling
    $\mathbf{F}''_{\mathbf{i}}$ = Conv2d($\mathbf{F}'_{\mathbf{i}}$, kernel=1, stride=1, output_channel=1)
      // Concentrate info to single channel
    $\mathbf{logit_i}$ = Flatten($\mathbf{F}''_{\mathbf{i}}$) ;   // Use activations of
      units at different layers as logits for
      patches of different sizes
$\mathbf{logits}$ = Concat([$\mathbf{logit_i}$])
$\mathbf{P}_{\text{crop}}$ = Normalize(Exponential($\mathbf{logits}$))
```

C.2. Test-time augmentation

Besides augmenting data during training, the learned invariance can also be applied to test-time augmentation. Given a test image $\mathbf{x}$, we sample n different transformations τ_i from $p(\tau; \phi(\mathbf{x}))$ and apply them to $\mathbf{x}$ to generate n different views $\tau_i(\mathbf{x})$. After feeding these views to the classifier, f, we use the mean logit $\frac{1}{n}\sum_{i=1}^{n} f(\tau_i(\mathbf{x}))$ to predict $\mathbf{x}$'s label. When only learning invariance for test-time augmentation, InstaAug can be trained with a fixed pre-trained classifier at a lower computation cost.

C.3. Other parametrization methods

Besides the uniform and location-related parameterization, we also tried VAE-like methods to parameterize augmentations, such as cropping. The main idea is to have a Gaussian latent variable and a neural decoder to map the latent Gaussian distributions to a continuous distribution on transformation parameters (in this case, the centers and sizes of crops). However, similar to the uniform parameterization, we find the VAE-like parameterization unstable and easily stuck at local minima.

C.4. Network structures

In all of our experiments, we use PreActResNet-18 as the base structure of ϕ for both uniform and location-related parameterizations. When dealing with multiple transformations, for example in Section 5.4, we use the same network to generate parameters for all transformations simultaneously.

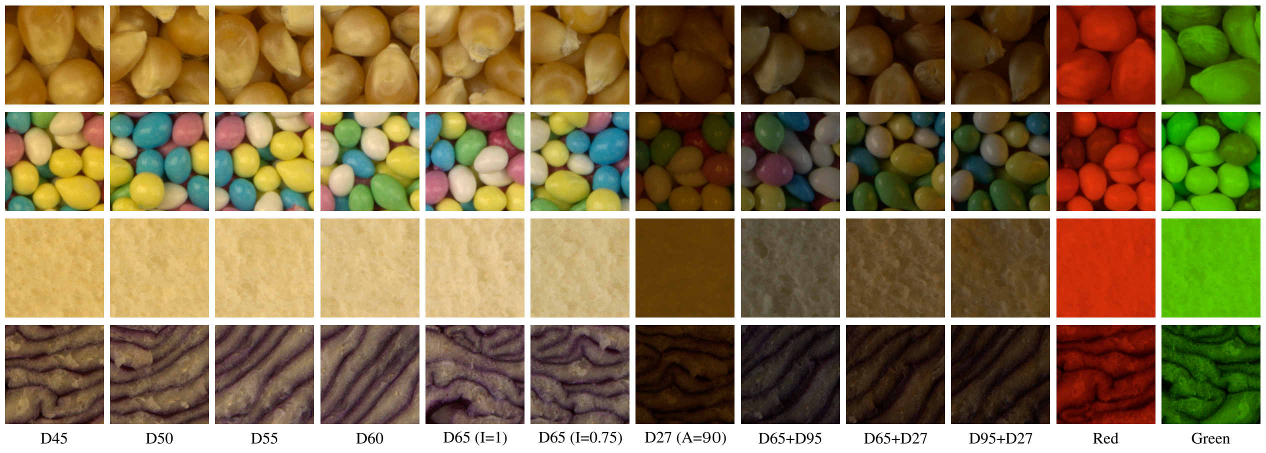


Figure C.1: Examples of RawFooT data. Each row contains images in the same class (corn, candies, floor, red cabbage) under different lighting conditions. The left and right half of lighting conditions are in the easy and hard groups, respectively.

D. Experimental details

D.1. Cropping

Supervised training Based on the Mixmo codebase[4] (Ramé et al., 2021), we use stochastic gradient descent (SGD) optimizer to train baselines and InstaAug. For the classifier, the initial learning rate is set to 0.2 (with momentum 0.9 and weight decay $1e-4$). A scheduler is used to decrease the learning rate by a factor of 0.9 once validation accuracy doesn't increase for 10 epochs. The learning rate of the augmentation module ϕ is fixed at $1e-5$. Batch size is set to 100 and we pre-train InstaAug for 10 epochs without augmentation. We train the model until convergence and the maximum epoch is set to 150.

Contrastive training We directly apply InstaAug on the codebase[5] from (Ermolov et al., 2021). Because of the characteristics of contrastive learning, we set the batch size to 512. Same as the supervised case, we use SGD optimizer to train the augmentation module ϕ. Differently, we use Adam optimizer (Kingma & Ba, 2015) (with learning rate $1e-3$ and weight decay $1e-6$) to train the base model. We train each model for 500 epochs and decrease the learning rate by a factor of 0.8 at step 450 and 475.

Implementation of LRP As an example, we show how to implement location-related parameterization with a basic CNN structure in Algorithm 1.

D.2. Color jittering on textures

Training. We use PreActResNet-18 ($width = 1$) on texture recognition task on RawFooT and train it with SGD optimizer. The learning rate is 0.02 (with momentum 0.9 and weight decay $1e-4$) for the classifier and $1e-5$ for the augmentation module ϕ. We train each model for 50 epochs and learning rate schedulers are not necessary in this task.

Table D.1: Splitting of Lighting conditions.

Group	Lighting id
Easy (1)	1-4,10,14-31
Hard (2)	5-9, 11-13, 32-46

Random augmentation baseline. We sweep over the variation range on each channel to find the best hyperparameters for the random augmentation baseline. For hue (h-jittering), we sweep between $[0, 0.5]$ with stride 0.1, and for saturation (s-jittering) as well as brightness value (v-jittering), we sweep between $[0, 1.0]$ with stride 0.2, which yields 216 different settings in total. The best accuracy shown in Table 3 is achieved where h,s,v$= 0.0, 0.2, 0.8$.

In-distribution vs. out-of-distribution generalization. To further investigate the effect of each augmentation method, we additionally split the 46 test sets into two equally-sized groups. The first group contains lighting conditions similar to D45, such as daylight with different temperatures, for which the vanilla model without augmentation trained on D45 has high test accuracy. The second group contains lighting conditions that are dramatically different from D45, for example, pure red light, which are more difficult for the vanilla method. Then the average accuracy on the first group can be regarded as a measure of in-distribution generalization, while the accuracy on the second group reflects out-of-distribution generalization.

D.3. Time complexity

On a single 1080Ti, each iteration of training InstaAug on TinyIN takes 0.25s, and each epoch takes 250s. As it takes

[4]https://github.com/alexrame/mixmo-pytorch.git, under Apache License v2.0.

[5]https://github.com/htdt/self-supervised.git, under Apache License v2.0.

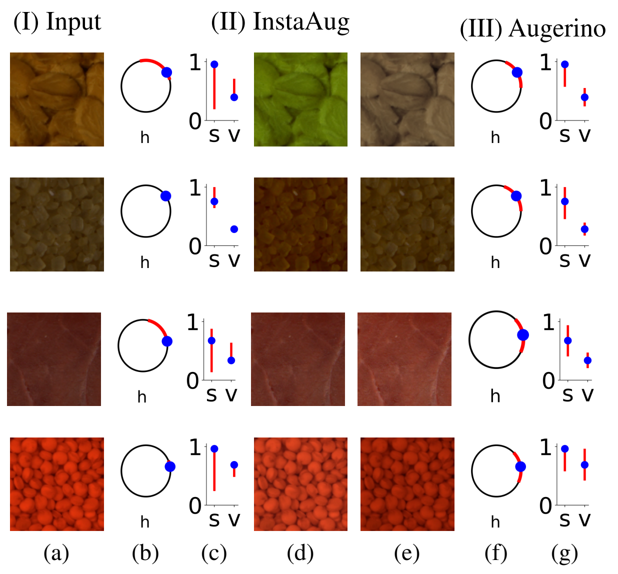


Figure D.1: Examples of learned color jittering. (a) Original image; (b, f) Average hue (H) of original image (blue dot) and learned hue jittering (red arc) for InstaAug and Augerino; (c,g) learned saturation (S) and brightness value (V) of original image (blue dot) and learned hue jittering (red line segment) for InstaAug and Augerino; (d,e) examples of images transformed by InstaAug.

150 epochs (1.5×10^5 iterations) for the model to converge, the total training time is about 10h. For random augmentation, each iteration takes 0.15s, and each epoch takes 150s. It takes the same 150 epochs (1.5×10^5 iterations) to converge, so the total training time is about 7h. All of the other augmentation learning approaches are slower than random augmentation, with some of them being noticeably slower than InstaAug itself. For example, the exploitation stage of AdaAug has the time complexity as random augmentation, which is 0.15s/iter, 150s/epoch, and 7h for the whole training process. However, its exploration took us more than 15h because it requires averaging the representations of a large number of augmented samples.

The training speed of InstaAug on RawFoot (color jittering) is similar to random augmentation (0.37s/iter vs. 0.40s/iter), though it takes more epochs (about 40) compared with random augmentation, which usually converges after 25 epochs.

E. Additional Results and Discussion

E.1. RawFooT

Figure D.1 shows some examples of learned color jittering. Though it's not easy to fully understand them, we can still find some patterns. For example, InstaAug tends to increase the brightness of darker images (row 1 and 3) and decrease the brightness of brighter images (row 4). Also, InstaAug is more likely to change saturation compared with hue and brightness, which is consistent with the common belief that saturation contains less information than hue and brightness.

InstaAug's behavior is quite different on different samples. It even decides not to augment the H and V channels of the image in the second row. In comparison, Augerino adds or multiplies noise to each channel with the same distribution across all samples, which is harmful in many cases. For example, the input image in the last row is already very bright. but Augerino allows further increasing its brightness. Then brightness values of many pixels will be capped at 1.0, which leads to loss of information.

Table E.1: Model performance with different choice of $H_{\min}$ and $H_{\max}$ on supervised cropping.

$H_{\min}$	$H_{\max}$	Accuracy (%)
0.0	0.5	52.12
0.5	1.0	61.28
1.0	1.5	62.91
1.5	2.0	64.39
2.0	2.5	65.04
2.5	3.0	65.05
3.0	3.5	**66.03**
3.5	4.5	65.60
4.0	4.5	64.35
4.5	5.0	64.17
0.0	1.0	51.78
1.0	2.0	63.96
2.0	3.0	65.25
3.0	4.0	**65.78**
4.0	5.0	64.23

E.2. Hyperparameter Ablation

The two hyperparameters of InstaAug are $H_{\min}$ and $H_{\max}$, which reflect human preference on augmentation diversity. To investigate how $H_{\min}$ and $H_{\max}$ influence model performance and provide a guide on how to choose them, we perform an ablation study for the experiment of Section 5.2, wherein we sweep over possible intervals of length 0.5 and 1.0. From Table E.1, we find that the best accuracy is achieved when $[H_{\min}, H_{\max}]$ is set to $[3, 3.5]$, while any sub-interval of $[2, 4]$ produces significantly better results compared with random augmentation.

To show the effect of dynamically tuning λ in InstaAug, we compare it with results with fixed λ in Figure E.1. We find that for small $\lambda \leq 0.1$, the entropy term is nearly 0 throughout training, which gives a result similar to no augmentation. For large $\lambda \geq 0.5$, the model suffers from

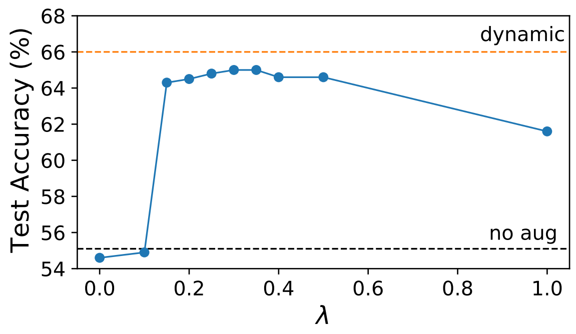


Figure E.1: Model performance with fixed λ on supervised cropping, compared with dynamic tuning λ used in InstaAug.

excessive augmentations throughout training which hinders the training of the classifier and InstaAug module. There is also a performance plateau for $0.15 \leq \lambda \leq 0.5$, whose accuracy is between 64.0 and 65.0. However, even the best of them is not as good as dynamic tuning, which is probably a result of their inability to keep transformation diversity stable during different stages of the training process.

E.3. Why is the Random Augmentation baseline so strong?

It is perhaps initially surprising that the Random Augmentation baseline in 5.2 is so strong compared to the other global augmentation schemes. In short, this occurs because the extensive hyperparameter sweep used for it turns out to be a more effective tuning mechanism than directly training global parameters simultaneously to the model. To be more precise, for any *global* cropping scheme (which includes random crop, Augerino, and InstaAug without input), there is little to be gained from using a non-uniform distribution on the position of the crops. As such, the only thing that can be usefully learned is the distribution on the *size* of the crops themselves. For the random crop baseline, we do an exhaustive sweep to establish the best distribution on crop sizes, meaning that this baseline represents a near-optimal global cropping augmentation. By comparison, InstaAug (without input) must still learn the optimal cropping size distribution during training, and the results suggest that it does not always manage to do this perfectly, tending to prefer under-diverse transformations. This is perhaps not surprising, as it does not have access to a validation set, unlike the hyperparameter sweep implicitly being deployed for the random crop baseline. The problem is seen even more starkly for Augerino, where the lack of LRP causes training to become stuck in highly sub-optimal local optima that yield very little transformation diversity.