PREPARE FOR THE WORST: GENERALIZING ACROSS DOMAIN SHIFTS WITH ADVERSARIAL BATCH NORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Adversarial training is the industry standard for producing models that are robust to small adversarial perturbations. However, machine learning practitioners need models that are robust to other kinds of changes that occur naturally, such as changes in the style or illumination of input images. Such changes in input distribution have been effectively modeled as shifts in the mean and variance of deep image features. We adapt adversarial training by adversarially perturbing these feature statistics, rather than image pixels, to produce models that are robust to distributional shifts. We also visualize images from adversarially crafted distributions. Our method, Adversarial Batch Normalization (AdvBN), significantly improves the performance of ResNet-50 on ImageNet-C (+8.1%), Stylized-ImageNet (+6.7%), and ImageNet-Instagram (+3.9%) over standard training practices. In addition, we demonstrate that AdvBN can also improve generalization on semantic segmentation.

ImageNet	ImageNet-Instagram	ImageNet-C	Stylized-ImageNet	ImageNet-AdvBN
99.9% goldfinch	**89.6% goldfinch**	**57.4% goldfinch**	**16.2% gong**	**10.3% hen-of-the-woods**
0.05% bulbul	7.4% sulphur butterfly	11.8% brambling	8.8% bolete	5.1% Ibizan hound
0.02% house finch	0.5% hummingbird	8.8% guillotine	4.5% fox squirrel	4.0% flamingo

Figure 1: **Images from ImageNet variants along with classification scores by a pre-trained ResNet-50 model**. The right-most image is generated by our Adversarial Batch Normalization module. Details on how we generate this image can be found in Section 3.

1 INTRODUCTION

Robust optimization for neural networks has been a major focus of recent research. A mainstream approach to reducing the brittleness of classifiers is *adversarial training*, which solves a min-max optimization problem in which an adversary makes perturbations to images to degrade network performance, while the network adapts its parameters to resist degradation (Goodfellow et al., 2015; Kurakin et al., 2017; Madry et al., 2018). The result is a hardened network that is no longer brittle to small perturbations to input pixels. While adversarial training makes networks robust to adversarial perturbations, it does not address other forms of brittleness that plague vision systems. For example, shifts in image style, lighting, color mapping, and domain shifts can still severely degrade the performance of neural networks (Hendrycks & Dietterich, 2019).

We propose adapting adversarial training to make neural networks robust to changes in image style and appearance, rather than small perturbations at the pixel level. We formulate a min-max game in which an adversary chooses *adversarial feature statistics*, and network parameters are then updated to resist these changes in feature space that correspond to appearance differences of input images. This

game is played until the network is robust to a variety of changes in image space including texture, color, brightness, *etc*.

The idea of adversarial feature statistics is inspired by the observation that the mean and variance of features maps encode style information, and thus, they enable the transfer of style information from a source image to a target image through normalization (Huang & Belongie, 2017; Ulyanov et al., 2016). Unlike standard approaches that rely on feature statistics from auxiliary images to define an image style, we use adversarial optimization of feature statistics to prepare classifiers for the worst-case style that they might encounter.

We propose training with *Adversarial Batch Normalization* (AdvBN) layers. Before each gradient update, the AdvBN layers perform an adversarial feature shift by re-normalizing with the most damaging mean and variance. By using these layers in a robust optimization framework, we create networks which are resistant to any domain shift caused by feature statistics shift. An advantage of this method is that it does not require additional auxiliary data from new domains. We show that robust training with AdvBN layers hardens classifiers against changes in image appearance and style using a range of vision tasks including Stylized-ImageNet (Geirhos et al., 2019) and ImageNet-Instagram (Wu et al., 2020).

2 BACKGROUND

2.1 FEATURE NORMALIZATION

Feature normalization is an important component of modern neural networks that stabilizes training and improves model generalization. Let $f \in \mathbb{R}^{N \times C \times H \times W}$ denote feature maps output by a layer, where N is the batch size, C is the number of channels, and H and W represent the height and width of the feature maps, respectively. Different normalization methods compute the mean, μ, and standard deviation, σ, over different dimensions of the feature maps. They use the derived feature statistics, often along with learned multiplicative and additive parameters, to produce normalized features, f':

$$f' = \gamma \cdot \frac{f - \mu(f)}{\sigma(f)} + \beta, \tag{1}$$

where γ and β are learnable parameters which re-scale and shift normalized features. For example, Batch Normalization (BN) (Ioffe & Szegedy, 2015) estimates feature statistics along the N, H, W dimensions. On the other hand, Instance Normalization (IN) (Ulyanov et al., 2016) computes μ and σ for each individual sample in the batch and only normalizes across the H and W dimensions.

Although feature normalization was originally proposed to accelerate the training process (Bjorck et al., 2018), previous work (Huang & Belongie, 2017; Li et al., 2017) has shown that feature statistics effectively capture information concerning the appearance of images. Motivated by this observation, we impose uncertainty on these statistics during training in order to obtain models that are less sensitive to non-semantic characteristics, thus generalizing to images with different appearances.

2.2 ADVERSARIAL TRAINING

Untargeted adversarial examples are generated by maximizing classification loss with respect to the input. One popular method, projected gradient descent (PGD), involves performing gradient ascent in the signed gradient direction and projecting the perturbation in order to enforce an ℓ_∞-norm constraint (Madry et al., 2018). Adversarial training aims to solve the saddlepoint optimization problem,

$$\min_\theta \mathbb{E}_{(X,y)\sim\mathcal{D}} \left[\max_{\|\delta\|_p < \epsilon} \mathcal{L}(g_\theta(X+\delta), y) \right], \tag{2}$$

where g_θ is a model with parameter vector θ, X, y is a clean input and the corresponding label drawn from distribution $\mathcal{D}$, and $\mathcal{L}$ denotes cross-entropy loss. Adversarial training solves this problem by iteratively sampling a batch of data, perturbing the batch adversarially, and performing a parameter update on the new adversarial batch (Madry et al., 2018). We harness adversarial training in order to create models robust to distributional shifts rather than the standard pixel-wise adversarial attacks.

3 ADVERSARIAL BATCH NORMALIZATION

We propose *Adversarial Batch Normalization* (AdvBN), a module that adversarially perturbs deep feature distributions such that the features confuse CNN classifiers. We iteratively compute adversarial directions in feature space by taking PGD steps on batch statistics. In the next section, we will train on these perturbed feature distributions in order to produce models robust to domain shifts.

Consider a pre-trained classification network, g, with L layers. We divide g into two parts, $g^{1,l}$ and $g^{l+1,L}$, where $g^{m,n}$ denotes layers m through n. Now, consider a batch of data, x, with corresponding labels, y. Formally, the AdvBN module is defined by

$$\text{BN}_{\text{adv}}(x; g, l, y) = \delta'_\sigma \cdot (f - \mu(f)) + \delta'_\mu \cdot \mu(f), \text{ where } f = g^{1,l}(x),$$

$$(\delta'_\mu, \delta'_\sigma) = \arg\max_{(\delta_\mu, \delta_\sigma)} \mathcal{L}\left[g^{l+1,L}\left(\delta_\sigma \cdot (f - \mu(f)) + \delta_\mu \cdot \mu(f)\right), y\right], \tag{3}$$

$$\text{subject to } \|\delta_\mu - 1\|_\infty \leq \epsilon, \|\delta_\sigma - 1\|_\infty \leq \epsilon,$$

where the maximization problem is solved with projected gradient descent. Simply put, the AdvBN module is a PGD attack on batch norm statistics which can be inserted inside a network. Note that $\delta_\mu, \delta_\sigma$ are vectors with length equal to the number of channels in the output of layer l, and we multiply by them entry-wise, one scalar entry per channel, similarly to Batch Normalization. Additionally, note that this module acts on a per-batch basis so that features corresponding to an individual image may be perturbed differently depending on the batch the image is in.

We formulate the perturbation by first subtracting out the mean so that $\delta_\mu \cdot \mu(f)$ is the new mean of the adversarial features, and δ_μ directly controls the new mean. We choose $\delta_\mu \cdot \mu(f)$, rather than simply δ_μ, to represent the new mean of the perturbed features so that ℓ_∞ bounds and steps size do not need to depend on the mean of f. Intuitively, we permit the mean of adversarial features to vary more when the clean features have a mean of higher magnitude.

Visualizing feature shifts To verify our assumption that adversarially perturbing feature statistics corresponds to transforming the distribution in image space, we visualize the effects of AdvBN. We adopt the VGG-19 based autoencoder from Huang & Belongie (2017). At the bottleneck of the autoencoder, we plug in an AdvBN module followed by a VGG-19 classifier (Simonyan & Zisserman, 2015). We visualize perturbations by feeding the AdvBN outputs to the decoder.

In Figure 2, images crafted through this procedure are visibly different from the originals; semantic content in the original images is preserved, but the new images exhibit differences in color, texture, and edges. We draw two major conclusions from these visualizations which highlight the adversarial properties of these domains. The first one concerns textures: according to Geirhos et al. (2019), CNNs rely heavily on image textures for classification. Images from the adversarial domain, on the other hand, have inconsistent textures across samples. For example, the furry texture of a dog is smoothed in column 2, and the stripes disappear from a zebra in column 4, whereas visible textures appear in columns 6 and 8. The second conclusion pertains to color. Results in Zhang et al. (2016)

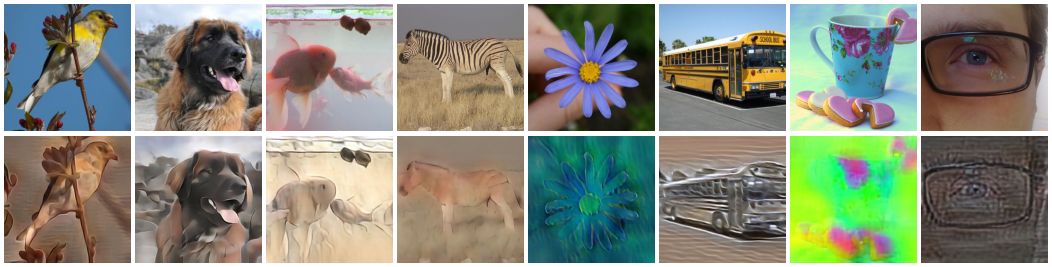

Figure 2: **Examples of perturbed ImageNet images generated by AdvBN with a decoder**. The first row contains the original versions from the ImageNet validation set.

suggest that colors serve as important information for CNNs. In the adversarial domain, we find suppressed colors (columns 1, 3) and unnatural hue (columns 5, 7). See Appendix A.3 for additional example images generated by this procedure. Figure 3 illustrates how the appearance of reconstructed images shifts as adversarial perturbations to feature statistics become larger.

We use this visualization technique to process the entire ImageNet validation set and denote it as ImageNet-AdvBN in Figure 1. By evaluating different methods on this dataset, we observe that accuracy on ImageNet-AdvBN is consistently lower than that on the original ImageNet dataset. This degradation in performance validates the adversarial property of features generated by AdvBN. Experiments concerning performance on ImageNet-AdvBN are listed in Appendix A.2.

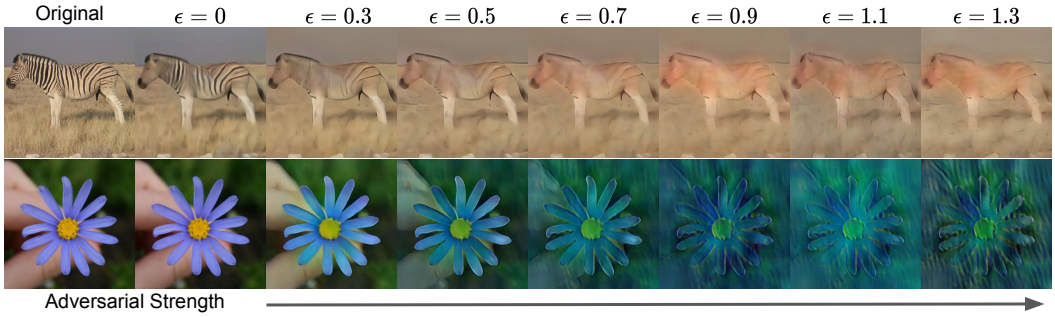

Figure 3: **The effect of adversarial strength on visualized examples**. $\epsilon = 0$ corresponds to images reconstructed by our autoencoder without AdvBN.

4 Training with Adversarial Batch Normalization

In this section, we use the proposed AdvBN module to train networks on the perturbed features. The goal of this adversarial training routine is to produce networks that both perform well on the data distribution on which they are trained and also generalize well to many new domains, all without having to obtain auxiliary data from new domains.

We start with a pre-trained model, $g = g^{l+1,L} \circ g^{1,l}$, and we fine-tune the subnetwork, $g^{l+1,L}$, on clean and adversarial features simultaneously. To this end, we solve the following min-max problem,

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}(g_\theta^{l+1,L} \circ \text{BN}_{\text{adv}} \circ g^{1,l}(x), y) + \mathcal{L}(g_\theta^{l+1,L} \circ g^{1,l}(x), y) \right], \tag{4}$$

where $\mathcal{L}$ denotes cross-entropy loss, and $\mathcal{D}$ is the distribution of batches of size n. This optimization problem contains a maximization problem inside the BN_{adv} layer. In order to maintain the network's performance on natural images, we adopt a similar approach to Xie et al. (2020) by using auxiliary batch normalization in $g^{l+1,L}$ for adversarial features; we use the original BNs when propagating clean features, and we use auxiliary ones for adversarial features. See Algorithm 1 for a detailed description of our method for fine-tuning the parameters of $g^{l+1,L}$.

Since we start with pre-trained models, we only need to fine-tune for 20 epochs, yielding improved robustness with little additional compute. Moreover, we only modify the parameters of later layers, so we do not need to backpropagate through the first half of the network. See Appendix B for an analysis of training times using our method. In the following section, we measure the performance, on several datasets, of our model fine-tuned using adversarial training with AdvBN.

5 Experiments

5.1 Implementation

Our method begins with a torchvision ImageNet pre-trained ResNet-50 (He et al., 2016). We insert the AdvBN module at the end of the 2nd convolutional stage. The model is fine-tuned following Algorithm 1 for 20 epochs. The learning rate starts at 0.001 and decreases by a factor of 10 after 10

Algorithm 1: Training with Adversarial Batch Normalization

Input: Training data, pretrained network $g = g_\theta^{l+1,L} \circ g^{1,l}$, PGD bound ϵ, and PGD step size τ
Result: Updated network parameters, θ, of subnetwork $g_\theta^{l+1,L}$
for *epoch = 1, ..., N* **do**
  Sample mini-batch x with label y;
  Obtain feature map $f = g^{1,l}(x)$;
  Initialize perturbation: $\delta = (\delta_\mu, \delta_\sigma)$;
  Let $f_{adv} = f$;
  for *adversarial step = 1, ..., m* **do**
    $f_{adv} \leftarrow \delta_\sigma \cdot (f - \mu(f)) + \delta_\mu \cdot \mu(f)$;
    Update δ: $\delta \leftarrow \delta + \tau \cdot sign(\nabla_\delta \mathcal{L}(g_\theta^{l+1,L}(f_{adv}), y))$;
    $\delta \leftarrow$ clip $(\delta, 1-\epsilon, 1+\epsilon)$;
  end
  $f_{adv} \leftarrow \delta_\sigma \cdot (f - \mu(f)) + \delta_\mu \cdot \mu(f)$;
  Minimize the total loss w.r.t. network parameter:
  $\theta \leftarrow \arg\min_\theta \mathcal{L}(g_\theta^{l+1,L}(f_{adv}), y) + \mathcal{L}(g_\theta^{l+1,L}(f), y)$;
end
return θ

epochs with a batch size of 256. We use SGD with momentum 0.9 and weight decay coefficient 10^{-4}. We augment inputs with a fixed AutoAugment (Cubuk et al., 2019) policy. Adversarial parameters are $\tau = 0.2$ and $\epsilon = 1.1$ with 6 repeats.

5.2 Generalization to ImageNet Variants

Datasets. We compare our method to other methods designed to produce classification networks which generalize better. The datasets we consider are variants of ImageNet (Deng et al., 2009):

- **ImageNet-C** (Hendrycks & Dietterich, 2019) contains distorted images with 15 categories of common image corruption applied, each with 5 levels of severity. Performance on this dataset is measured by mean Corruption Error (mCE), the average classification error over all 75 combinations of corruption type and severity level, weighted by their difficulty.
- **ImageNet-Instagram** (Wu et al., 2020) is composed of ImageNet images filtered with a total of 20 different Instagram filters. This dataset contains 20 versions of each ImageNet image, each with a different filter applied.
- **Stylized-ImageNet** (Geirhos et al., 2019) consists of images from the ImageNet dataset, each stylized using AdaIN (Huang & Belongie, 2017) with a randomly selected painting. Textures and colors of images in this dataset differ heavily from the originals.

Models. Our baseline model is the publicly available torchvision ResNet-50 pre-trained on ImageNet, denoted as "Standard" in Table 1. All models we compare to, aside from SIN (Geirhos et al., 2019), are not trained on any of the ImageNet variants that are used for evaluation. The PGD model is adversarially trained with the PGD attack on inputs and is provided by Engstrom et al. (2019). MoEx (Li et al., 2020) changes feature moments inside networks but not in an adversarial manner. IBN-Net (Pan et al., 2018) improves the generalization of networks by combining batch normalization and instance normalization. AugMix (Hendrycks et al., 2020) is a data augmentation method that solves the distributional mismatch between training and testing data and increases classification robustness. SIN is a network trained on both Stylized ImageNet and ImageNet. We do not measure the accuracy of SIN on Stylized-ImageNet since it acquires knowledge of the target domain during training. Note that all models we use in our comparisons are the original versions released by the authors of the original work.

Results. As shown in Table 1, the performance of baseline model significantly degrades on all three ImageNet variants, highlighting the brittleness of this high performance classification model when

Table 1: **Performance of AdvBN and alternative methods on ImageNet variants**.

Method	ImageNet-C mCE ↓	ImageNet-Instagram top1/ top5 acc. ↑	Stylized-ImageNet top1/ top5 acc.↑
Standard	76.7	67.2/ 87.6	7.4/ 16.4
PGD	85.0	49.0/ 71.4	12.5/ 23.9
MoEx	74.8	70.0/ 89.4	5.0/ 12.0
IBN-Net	70.3	69.6/ 89.3	10.7/ 22.2
Augmix	**68.4**	70.4/ 89.4	11.2/ 23.1
SIN	69.3	66.9/ 87.4	–/ –
AdvBN	68.6	**71.1/ 89.5**	**14.1/ 26.9**

tested on novel distributions. Fine-tuning with AdvBN, on the other hand, substantially improves the performance of the standard ResNet-50 model. In particular, we achieve an 8.1% accuracy gain on ImageNet-C through fine-tuning with AdvBN. On Stylized-ImageNet and ImageNet-Instagram, our model not only vastly improves upon the baseline model but also achieves the best performance among all methods with which we compare. The consistent performance boost across all three benchmarks demonstrates that adversarial training with AdvBN can effectively enhance robustness against various distributional shifts. Note that this AdvBN model has additional auxiliary BN layers, so its performance on the original ImageNet dataset is well maintained, as will be shown in Table 2 in the next subsection. See Appendix C for details concerning inference with auxiliary BN layers.

5.3 Ablation Study

Where should the AdvBN module be placed within a network? The proposed AdvBN module can be inserted after any layer in a deep network. In this part, we try inserting the AdvBN at deeper layers, *i.e.*, `conv3_4` and `conv4_6`. From the results in Table 2, we observe that `conv4_6` yields the worst performance among all three ImageNet variants, indicating that using AdvBN at deeper layers is not as helpful as at shallower layers. We hypothesize two possible explanations for this phenomenon: (1) there are fewer trainable parameters when only very deep layers are fine-tuned; (2) features are more abstract in deeper layers, and perturbing the mean and standard deviation of high-level features can lead to extremely chaotic feature representations that are harmful for classification.

Table 2: **Ablation studies**.

Model	ImageNet top1 acc. ↑	ImageNet-C mCE. ↓	ImageNet-Instagram top1 acc. ↑	Stylized-ImageNet top1 acc. ↑
Standard	76.1	76.7	67.2	7.4
l = `conv2_3`	76.5	68.6	71.1	14.1
l = `conv3_4`	76.0	70.0	70.2	19.5
l = `conv4_6`	75.3	75.0	68.5	11.0
$\epsilon = 0.5$	76.5	69.4	70.8	13.3
$\epsilon = 0.7$	76.4	69.0	70.9	13.5
$\epsilon = 0.9$	76.6	69.0	71.0	13.8
$\epsilon = 1.1$	76.5	68.6	71.1	14.1
$\epsilon = 1.3$	76.4	68.3	70.7	13.2
$\epsilon = 1.5$	76.2	68.8	70.4	13.1
additive δ	75.9	69.4	70.2	13.7
AutoAugment*	76.4	72.1	70.1	8.2

Adversarial strength. The strength of the adversarial attack in the adversarial training framework has a major impact on model performance (Madry et al., 2018). We test a range of PGD parameters to demonstrate how the strength of AdvBN affects model performance. We measure strength by the perturbation bound ϵ, where we fix τ to be 0.2 for all settings, and change the number of repeats for different bounds. The number of repeats n for each ϵ is set to be $n = [\epsilon/0.2] + 1$. Results concerning the impact of adversarial strength are listed in Table 2.

Multiplicative vs. additive perturbations. We originally chose a multiplicative noise so that a single perturbation bound can be applied to various layers and architectures regardless of the range in which feature values lie. To adjust the perturbation bound for an additive noise for each batch of feature, we compute maximum mean and standard deviation values across channels, μ_{max} and σ_{max}. Then, in the projection step of the PGD attack, we project perturbations to the mean and standard deviation into the ranges $(-\epsilon \cdot \mu_{max}, \epsilon \cdot \mu_{max})$ and $(-\epsilon \cdot \sigma_{max}, \epsilon \cdot \sigma_{max})$, respectively. The model is denoted "additive δ" in Table 2. This variant results in slightly degraded accuracy on each dataset.

Data augmentation. AdvBN performing in feature space can easily be combined with input space data augmentation. To determine what portion of the improvements we observed can be credited to AdvBN, we fine-tune a pre-trained ResNet-50 following the same routine as before but without the AdvBN module, adopting the same fixed AutoAugment policy along with all other hyperparameters. This method is denoted by AutoAugment* in Table 2. We see that fine-tuning with AutoAugment alone does not result in nearly as much improvement as the combined method on all datasets we consider; even performance on the original ImageNet dataset benefits from the AdvBN module.

5.4 Feature divergence analysis

We compare the features extracted by our network to those of a standard ResNet-50 trained on ImageNet. Following Pan et al. (2018), we model features from each channel using a normal distribution with the same mean and standard deviation, and we compute the symmetric KL divergence between the corresponding distributions on the two datasets (A and B). For two sets of deep features, F_A and F_B, each with C channels, the divergence $D(F_A||F_B)$ is computed using the formula,

$$\begin{aligned} D(F_A||F_B) &= \tfrac{1}{C}\textstyle\sum_{i=1}^{C}(KL(F_A^i||F_B^i) + KL(F_B^i||F_A^i)), \\ KL(F_A^i||F_B^i) &= \log \tfrac{\sigma_B^i}{\sigma_A^i} + \tfrac{{\sigma_A^i}^2 + (\mu_A^i - \mu_B^i)^2}{2{\sigma_B^i}^2} - \tfrac{1}{2}, \end{aligned} \tag{5}$$

where F^i denotes the features of i-th channel with mean μ^i and standard deviation σ^i.

In Figure 4, we compare the baseline model with our own on two pairs of datasets in the fine-tuned layers. Since ImageNet-Instagram contains 20 filter versions, we use the "Toaster" filter found in (Wu et al., 2020) to cause the sharpest drop in classification performance.

We find that the feature divergence in our network trained with AdvBN is substantially smaller on all layers in the fine-tuned subnetwork. In other words, the distribution of deep features corresponding to shifted domains is very similar to the distribution of deep features corresponding to standard ImageNet data. The small divergence between feature representations explains the effectiveness of AdvBN from a different angle and explains why our model generalizes well across datasets.

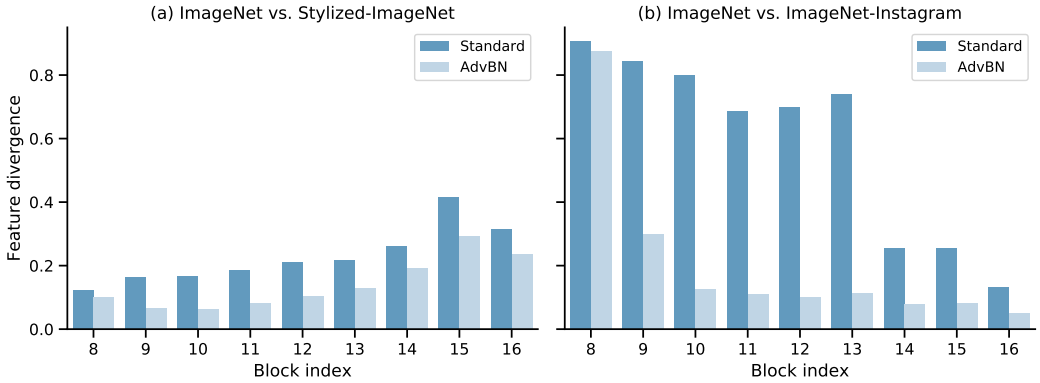

Figure 4: **Feature divergence between pairs of datasets using features extracted by AdvBN and a standard ResNet50**.

5.5 GENERALIZATION ON SEMANTIC SEGMENTATION

We now evaluate AdvBN in the context of semantic segmentation. We train a ResNet-50 based network with dilated convolutions (Yu & Koltun, 2016) on Cityscapes (Cordts et al., 2016) as our baseline model following the training protocol in Pan et al. (2018). Cityscapes comprises urban street scenes from various cities and contains 2975 images with fine annotations for training and 500 images for validation. To evaluate generalizability, we test the performance of the model on another dataset, GTA5 (Richter et al., 2016). The GTA5 dataset consists of synthetic images extracted from computer games, which have similar street scene content as Cityscapes and compatible label categories. The GTA5 validation set we use contains 6382 densely annotated images.

To apply AdvBN, we fine-tune the baseline model with AdvBN plugged in after layer `conv2_3` on Cityscapes for 20 epochs with adversarial training parameters $\tau = 0.15$, $\epsilon = 0.4$, and 4 repeats. In table 3, we observe a performance gain of 8.2% in mean IoU on the GTA5 dataset compared to the baseline model. The pixel accuracy also improves by 11.3%. Numbers on the left side of arrows denote performance on Cityscapes, and numbers on the right side of arrows denote performance on the GTA5 dataset.

Table 3: **Cityscapes** $\rightarrow$ **GTA5**.

	mean IoU (%)	Pixel Acc. (%)
Baseline	72.0 $\rightarrow$ 30.6	95.3 $\rightarrow$ 68.5
AdvBN	71.5 $\rightarrow$ 38.8	95.2 $\rightarrow$ 79.8

6 RELATED WORK

Adversarial training Adversarial training and its variants (Goldblum et al., 2020; Madry et al., 2018; Shafahi et al., 2019) have been widely studied for producing models that are robust to adversarial examples (Moosavi-Dezfooli et al., 2016; Szegedy et al., 2014). Recent work considers adversarial training as data augmentation (Tsipras et al., 2019). Xie et al. (2020) finds that deep features corresponding to adversarial examples have different mean and standard deviation than those corresponding to natural images. This work takes advantage of the distributional discrepancy to improve performance on non-adversarial data. Our work also adopts the adversarial training framework to make models robust against other kinds of perturbations, but instead of crafting adversarial examples in image space, we craft adversarial feature distributions by perturbing feature statistics.

Robustness to distributional shifts While extensive effort has been made to improve the robustness of classifiers to adversarial examples, there are other kinds of robustness that deep neural networks must address in order for them to be reliable. Corrupted images and new domains pose major challenges to networks with standard training (Geirhos et al., 2019; Hendrycks & Dietterich, 2019; Wu et al., 2020). Performance degradation on these images can be attributed to shifts in data distributions (Gilmer et al., 2018). In order to produce networks which generalize well, one common practice is to perform data augmentation (Cubuk et al., 2019; Hendrycks et al., 2020; Yun et al., 2019). However, the benefits of data augmentation are largely limited by the types of augmentations used during training (Geirhos et al., 2018). Feature space augmentation (Li et al., 2020) replaces feature statistics corresponding to one sample with ones corresponding to another sample. Our work can be also considered as feature space augmentation, we instead consider a worst-case scenario in the context of feature space distributional shifts by adopting the adversarial training framework.

7 CONCLUSION

Our work studies how adversarially perturbing feature statistics simulates distributional shift in image data. We find that adversarial fine-tuning on features perturbed in this way improves robustness to data stylization and corruption without ever training on auxiliary data. Training with Adversarial Batch Normalization (AdvBN) is computationally cheap and can quickly make pre-trained models less brittle. We fine-tune a ResNet-50 with our algorithm and surpass the performance of state-of-the-art methods on both ImageNet-Instagram and Stylized-ImageNet. Adversarial feature statistics are a promising direction for creating models that generalize well to a variety of domains.

REFERENCES

Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. In *NIPS*, 2018.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019. URL `https://github.com/MadryLab/robustness`.

Robert Geirhos, Carlos RM Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge, and Felix A Wichmann. Generalisation in humans and deep neural networks. In *NIPS*, 2018.

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *ICLR*, 2019.

Justin Gilmer, Ryan P. Adams, Ian J. Goodfellow, David Andersen, and George E. Dahl. Motivating the rules of the game for adversarial example research. *CoRR*, abs/1807.06732, 2018.

Micah Goldblum, Liam Fowl, Soheil Feizi, and Tom Goldstein. Adversarially robust distillation. In *AAAI*, 2020.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *ICLR*, 2020.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *ICLR*, 2017.

Boyi Li, Felix Wu, Ser-Nam Lim, Serge J. Belongie, and Kilian Q. Weinberger. On feature normalization and data augmentation. *CoRR*, abs/2002.11102, 2020.

Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. In *ICLR Workshop*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016.

Xingang Pan, Ping Luo, Jianping Shi, and Xiaoou Tang. Two at once: Enhancing learning and generalization capacities via ibn-net. In *ECCV*, 2018.

Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *ECCV*, 2016.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *NeurIPS*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *ICLR*, 2019.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

Zhe Wu, Zuxuan Wu, Bharat Singh, and Larry S. Davis. Recognizing instagram filtered images with feature de-stylization. In *AAAI*, 2020.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L. Yuille, and Quoc V. Le. Adversarial examples improve image recognition. In *CVPR*, 2020.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

A IMAGENET-ADVBN EXPERIMENTS

A.1 CREATION OF THE IMAGENET-ADVBN DATASET

We process the entire ImageNet validation set using the visualization technique introduced in Section 3. We consider two encoder architectures: one is the VGG-19 encoder we use for visualization, another consists of layers of a ResNet-50 up to `conv2_3`. Both encoders are paired with the same decoder architecture from Huang & Belongie (2017). The resulting datasets, denoted by ImageNet-AdvBN-VGG and ImageNet-AdvBN-ResNet respectively, contain 50000 images each. The data we synthesize for testing other models is generated using these autoencoders that contain the AdvBN module but on ImageNet validation data. AdvBN is conducted with 6 steps, stepsize $= 0.20$, $\epsilon = 1.1$, and a batchsize of 32. We do not shuffle the ImageNet validation data when generating these batches.

A.2 CLASSIFICATION ON IMAGENET-ADVBN

Table 4: **Classification performance on ImageNet and ImageNet-AdvBN**.

Method	ImageNet top1 acc. ↑	Im-Adv-VGG top1/ top5 acc. ↑	VGG Reconstructed top1/ top5 acc. ↑	Im-Adv-ResNet top1/ top5 acc. ↑	ResNet Reconstructed top1/ top5 acc. ↑
Standard	76.1	1.6/ 4.7	45.8/ 70.6	0.4/ 1.3	65.7/ 86.9
PGD	62.4	15.4/ 30.2	54.7/ 77.8	1.0/ 2.2	61.2/ 83.1
MoEx	79.1	1.0/ 2.9	40.2/ 63.8	0.3/ 1.1	65.7/ 86.8
IBN-Net	77.2	2.3/ 6.2	49.2/ 73.9	0.6/ 1.9	69.4/ 89.4
AugMix	77.6	3.9/ 9.9	53.5/ 77.0	1.0/ 2.7	71.9/ 90.7
SIN	74.6	10.9/ 24.4	51.7/ 75.8	1.7/ 4.3	67.2/ 87.9
Ours	76.5	6.3/ 15.6	54.6/ 78.3	2.9/ 6.4	66.4/ 87.3

Table 4 shows the classification performance of various models on the two ImageNet-AdvBN variants, denoted as Im-Adv-VGG and Im-Adv-ResNet respectively. We also test these models on ImageNet images that are reconstructed using our autoencoders, denoted as VGG Reconstructed and ResNet Reconstructed, for each autoencoder. The performance gap between ImageNet-AdvBN and Reconstructed ImageNet indicates that the degradation on ImageNet-AdvBN is not solely caused by the reconstruction loss due to the autoencoders we use.

A.3 ADDITIONAL EXAMPLE IMAGES

We include more images from ImageNet-AdvBN-VGG in this section. Example images in Figure 5 are randomly chosen. We do not include the ImageNet-AdvBN-ResNet, because the resulting images are mostly in extreme contrast with small textures that are hard to observe. It is possible that features output from ResNet based encoders are more sensitive to AdvBN perturbations; another explanation is that the features we extract from ResNet-50 are relatively shallow features compared to their VGG counterparts.

B RUNTIME ANALYSIS

B.1 RUNTIME OF TRAINING WITH ADVBN

We evaluate the training time of our method on a workstation with 4 GeForce RTX 2080 Ti GPUs. We use the default settings for AdvBN on ResNet-50: an AdvBN module after the `conv2_3` layer, a fixed AutoAugment policy, and 20 epochs of fine-tuning with 6-step PGD inside the AdvBN module. Fine-tuning is conducted on the ImageNet training set, containing 1.3 million images. Training in this setting takes approximately 40 hours with batchsize set to 256 .

B.2 COMPARISON WITH OTHER METHODS CONCERNING TRAINING BUDGET

We use the same infrastructure above to evaluate the training time of other ResNet-50 models that appear in Table 1. Training code for all other methods are obtained from official repository. For all methods, we use a batch size of 128, because augmix(Hendrycks et al., 2020) cannot run with

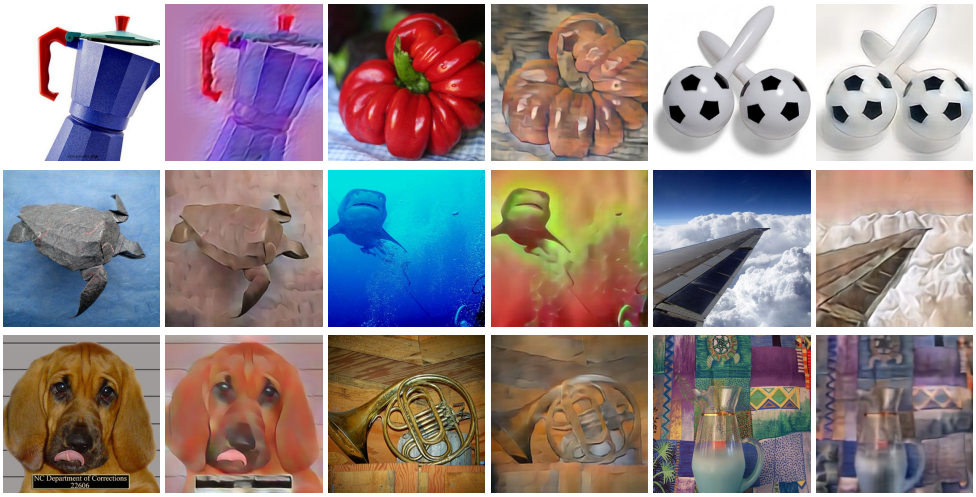

Figure 5: More example images. For each pair of adjacent columns, original versions are on the left, ImageNet-AdvBN-VGG is on the right.

a batch size of 256 on our workstation due to limited GPU memory. The number of processes in the dataloader is set to be 16. The speed values in Table 5 are averaged over 100 iterations. The estimated training duration is calculated by multiplying the speed and corresponding total number of iterations. Time spent on evaluation after each epoch is not considered in this estimation.

Table 5: **Runtime Analysis**.

Model	Speed (seconds/ iter.)	Epochs	Estimated training duration (hours)
Standard	0.17	90	43
PGD	2.95	90	738
MoEx	0.19	300	168
IBN-Net	0.22	100	61
AugMix	0.67	180	335
SIN	0.34	45	43
AdvBN	0.83	20	46

Training with AdvBN takes a long time per iteration because each iteration contains 6 PGD steps. The SIN (Geirhos et al., 2019) runtime speed is estimated based on standard training, since the model architectures and training procedures of these two methods are the same, except that SIN is trained for half the epochs but twice the data and thus twice the number of iterations per epoch. Training SIN requires additional access to Stylized-ImageNet as training data, which takes 134GB disk space; the time for generating the Stylized-ImageNet dataset and the extra storage cost are not considered in Table 5.

C INFERENCE USING MODELS TRAINED WITH ADVBN

Models containing Batchnorm layers will have two set of BN statistics in deeper layers that have been fine-tuned by AdvBN because we use auxiliary BNs introduced by (Xie et al., 2020) for propagating adversarial features crafted by the AdvBN module. During evaluation, we can choose either of the BN statistics to normalize features. The results we report in previous sections with regard to ImageNet, ImageNet-C and ImageNet-Instagram are obtained by using BN statistics corresponding to original features. We only use auxiliary BNs, which keep the batch statistics of adversarial features, to test performance on Stylized-ImageNet in Table 1. We also use auxiliary BNs for evaluating performances on ImageNet-AdvBN and Reconstructed ImageNet in Table 4.