On Feature Normalization and Data Augmentation

Boyi Li∗12  Felix Wu∗3  Ser-Nam Lim4  Serge Belongie12  Kilian Q. Weinberger13
1Cornell University  2Cornell Tech  3ASAPP  4Facebook AI
{bl728,sjb344,kilian}@cornell.edu  fwu@asapp.com  sernamlim@fb.com

Abstract

The moments (a.k.a., mean and standard deviation) of latent features are often removed as noise when training image recognition models, to increase stability and reduce training time. However, in the field of image generation, the moments play a much more central role. Studies have shown that the moments extracted from instance normalization and positional normalization can roughly capture style and shape information of an image. Instead of being discarded, these moments are instrumental to the generation process. In this paper we propose Moment Exchange, an implicit data augmentation method that encourages the model to utilize the moment information also for recognition models. Specifically, we replace the moments of the learned features of one training image by those of another, and also interpolate the target labels—forcing the model to extract training signal from the moments in addition to the normalized features. As our approach is fast, operates entirely in feature space, and mixes different signals than prior methods, one can effectively combine it with existing augmentation approaches. We demonstrate its efficacy across several recognition benchmark data sets where it improves the generalization capability of highly competitive baseline networks with remarkable consistency.

1. Introduction

Image recognition and image generation are two cornerstones of computer vision. While both are burgeoning fields, specialized techniques from both sub-areas can sometimes form a dichotomy. Examples are mixup [74] and squeeze-and-excitation [21] from the former, and adaptive instance normalization [24] from the latter, although exceptions exist. Historically, the field of deep learning was widely popularized in discriminative image classification with AlexNet [34], and image generation through GANs [14] and VAEs [32].

One particular aspect of this dichotomy is that in image recognition, popularized by batch normalization [26], the first and second moments (a.k.a., mean and standard deviation) in image recognition are computed across instances in a mini-batch and typically removed as noise [26, 65]. Studies have shown that this smoothes the optimization landscape [50] and enables larger learning rates [2], which leads to faster convergence in practice. In contrast, for techniques like instance normalization [59] and positional normalization [36], moments play a central role in the image generation process. For example, exchanging moments of latent features across samples has become a popular way to control for the style or shape of generated images [24, 25, 29, 36, 45]. Here, moments are viewed as features, not noise, with research showing that they encode the style of an image [24, 29], as well as the underlying structure [36]. To substantiate this point, we depict the first and second moments of the features extracted in the first layer of a ResNet [18] in Fig. 1, using the technique described in [36]. The class label can still be inferred visually from both moments, which is a testament to the signal that remains in these statistics. To further substantiate our observation, we also show in Fig. 2 that simply using moments (from the first ResNet layer) for image classification already yields non-trivial performance (red bar) as compared to random guessing (gray bar). Similarly, removing the moments from positional normalization has a detrimental effect (blue bar vs. green bar).

As there is evidently important signal in the moments and in the normalized features, we would like to introduce a

∗: Equal contribution.

Example images are from Shutterstock.
way to regulate how much attention the deep net should pay to each source. One approach to direct neural networks to a particular signal source is to introduce dedicated feature augmentation. For example, it has been shown that ConvNets trained on ImageNet [7] are biased towards textures instead of shapes [12]. To overcome this, Geirhos et al. [12] introduce a style transfer model to create a set of images with unreal textures. For example, they generate cats with elephant skins and bears with Coca-Cola bottle texture. An image classifier is trained to recognize the shape (cats or bears) instead of the textures (elephants or bottles).

In this paper we propose a novel data augmentation scheme that, to our knowledge, is the first method to systematically regulate how much attention a network pays to the signal in the feature moments. Concretely, we extract the mean and variance (across channels) after the first layer, but instead of simply removing them, we swap them between images. See Fig. 3 for a schematic illustration, where we extract and remove the feature moments of a cat image, and inject the moments of a plane. Knowing that the resulting features now contain information about both images, we make the network predict an interpolation of the two labels. In the process, we force the network to pay attention to two aspects of the data: the normalized feature (from the cat) and the moments (from the plane). By basing its prediction on two different signals we increase the robustness of the classification, as during testing both would point towards the same label. We call our method Moment Exchange or MoEx for short.

Through exchanging the moments, we swap the shape (or style) information of two images, which can be viewed as an implicit version of the aforementioned method proposed by Geirhos et al. [12]. However, MoEx does not require a pre-trained style transfer model to create a dataset explicitly. In fact, MoEx is very effective for training with mini-batches and can be implemented in a few lines of code: During training we compute the feature mean and variance across the entire input, or a randomly picked mini-batch (as it is the case for BatchNorm), and permutate them across the mini-batch, and re-inject them into the feature representation of other instances (while interpolating the labels).

MoEx operates purely in feature space and can therefore easily be applied jointly with existing data augmentation methods that operate in the input space, such as cropping, flipping, rotating, but even label-perturbing approaches like Mixup [74] or Cutmix [72]. Importantly, because MoEx only alters the first and second moments of the pixel distributions, it has an orthogonal effect to existing data augmentation methods and its improvements can be “stacked” on top of their established gains in generalization. We conduct extensive experiments on eleven different tasks/datasets using more than ten varieties of models. The results show that MoEx consistently leads to significant improvements across models and tasks, and it is particularly well suited to be combined with existing augmentation approaches. Further, our experiments show that MoEx is not limited to computer vision, but is also readily applicable and highly effective in applications within speech recognition and natural language processing—suggesting that MoEx reveals a fundamental insight about deep nets that crosses areas and data types.

2. Background and Related Work

Feature normalization has always been a prominent part of neural network training [35, 37]. Initially, when networks had predominately one or two hidden layers, the practice of z-scoring the features was limited to the input itself. As networks became deeper, Ioffe and Szegedy [26] extended the practice to the intermediate layers with the celebrated BatchNorm algorithm. As long as the mean and variance are computed across the entire input, or a randomly picked mini-batch (as it is the case for BatchNorm), the extracted moments reveal biases in the data set with no predictive information — removing them causes no harm but can substantially improve optimization and generalization [2, 35, 49].

In contrast, recently proposed normalization methods [1, 36, 59, 65] treat the features of each training instance as a distribution and normalize them for each sample individually. We refer to the extracted mean and variance as intra-instance moments. We argue that intra-instance moments are attributes of a data instance that describe the distribu-
We denote this function as asymmetric composition, as it mixes two very different components: The normalized features of one instance are combined with the features of another. This asymmetric composition in feature space allows us to capture and smooth out different directions of the decision boundary, not previously covered by existing augmentation approaches. We also show that MoEx can be implemented very efficiently in a few lines of code, and should be regarded as an effective default companion to existing data augmentation methods.

**Setup.** Deep neural networks are composed of layers of transformations including convolution, pooling, transformers [61], fully connected layers, and non-linear activation layers. Consider a batch of input instances $x$, these transformations are applied sequentially to generate a series of hidden features $h^1, ..., h^L$ before passing the final feature $h^L$ to a linear classifier. For each instance, any feature presentation $h^i$ is a 3D tensor indexed by channel (C), height (H), and width (W).

**Normalization.** We assume the network is using an invertible intra-instance normalization. We denote this function by $F$, which takes the features $h^i$ of the $i$-th input $x_i$ at layer $\ell$ and produces three outputs: the normalized features $\hat{h}^i$, the first moment $\mu^i$, and the second moment $\sigma^i$:

$$ (\hat{h}^i, \mu^i, \sigma^i) = F(h^i), \quad h^i = F^{-1}(\hat{h}^i, \mu^i, \sigma^i). $$

The inverse function $F^{-1}$ reverses the normalization process. As an example, PONO [36] computes the first and second moments across channels from the feature representation at a given layer

$$ \mu_{b,h,w}^\ell = \frac{1}{C} \sum_c h_{b,c,h,w}^\ell, $$

$$ \sigma_{b,h,w}^\ell = \sqrt{\frac{1}{C} \sum_c \left( h_{b,c,h,w}^\ell - \mu_{b,h,w}^\ell \right)^2} + \epsilon. $$

The normalized features have zero-mean and standard deviation 1 along the channel dimension. Note that after using MoEx with an *inter-instance* normalization to exchange features, we can still apply an *inter-instance* normalization (like BatchNorm) on these exchanged or mixed features.

3. Moment Exchange

In this section we introduce Moment Exchange (MoEx), which blends feature normalization with data augmentation. Similar to Mixup and Cutmix, it fuses features and labels across two training samples, however it is unique in its asymmetry, as it mixes two very different components: The normalized features of one instance are combined with the feature moments of another. This asymmetric composition of its features and should not be discarded. Recent works [24, 36] have shown that such attributes can be useful in several generative models. Realizing that these moments capture interesting information about data instances, we propose to use them for data augmentation.

Data augmentation has a similarly long and rich history in machine learning. Initial approaches discovered the concept of label-preserving transformations [51, 53] to mimic larger training data sets to suppress overfitting effects and improve generalization. For instance, Simard et al. [54] randomly translate or rotate images assuming that the labels of the images would not change under such small perturbations. Many subsequent papers proposed alternative flavors of this augmentation approach based on similar insights [5, 6, 8, 29, 30, 38, 55, 68, 76]. Beyond vision tasks, back-translation [3, 9, 52, 71] and word dropout [27] are commonly used to augment text data. Besides augmenting inputs, Ghiasi et al. [13], van der Maaten et al. [60], Wang et al. [62] adjust either the features or loss function as implicit data augmentation methods. In addition to label-perturbing transformations, there is an increasing trend to use label-perturbing data augmentation methods. Zhang et al. [74] arguably pioneered the field with Mixup, which interpolates two training inputs in feature and label space simultaneously. Cutmix [72], instead, is designed especially for image inputs. It randomly crops a rectangular region of an image and pastes it into another image, mixing the labels proportional to the number of pixels contributed by each input image to the final composition.

![Figure 3. MoEx with PONO normalization. The features $h_A$ of the cat image are normalized and then infused with moments $\mu_B, \sigma_B$ from the plane image. See Appendix for more examples.](image)

$\mu_A, \sigma_A$ are the mean and standard deviation of the input features $h_A$. $\mu_B, \sigma_B$ are the mean and standard deviation of the input features $h_B$. MoEx takes the features $h_A$ and $h_B$ and applies a linear transformation to produce the final features $h_{\text{MoEx}}$.
with their well-known beneficial impact on convergence. As the norms compute statistics across different dimensions their interference is insignificant.

**Moment Exchange.** The procedure described in the following functions identically for each layer it is applied to and we therefore drop the $\ell$ superscript for notational simplicity. Further, for now, we only consider two randomly chosen samples $x_A$ and $x_B$ (see Fig. 3 for a schematic illustration). The intra-instance normalization decomposes the features of input $x_A$ at layer $\ell$ into three parts, $\hat{h}_A, \mu_A, \sigma_A$. Traditionally, batch-normalization [26] discards the two moments and only operates with the normalized features $\hat{h}_A$. If the moments are computed across instances (e.g. over the mini-batch) this makes sense, as they capture biases that are independent of the label. However, in our case we focus on intra-instance normalization (See Fig. 1), and therefore both moments are computed only from $x_A$ and are thus likely to contain label-relevant signal. This is clearly visible in the cat and plane examples in Fig. 3. All four moments ($\mu_A, \sigma_A, \mu_B, \sigma_B$) capture the underlying structure of the samples, revealing their respective class labels.

We consider the normalized features and the moments as distinct views of the same instance. It generally helps robustness if a machine learning algorithm leverages multiple sources of signal, as it becomes more resilient in case one of them is under-expressed in a test example. For instance, the first moment conveys primarily structural information and only little color information, which, in the case of cat images can help overcome overfitting towards fur color biases in the training data set.

In order to encourage the network to utilize the moments, we use the two images and combine them by injecting the moments of image $x_B$ into the feature representation of image $x_A$: $h_A^{(B)} = F^{-1}(\hat{h}_A, \mu_B, \sigma_B)$. In the case of PONO, the transformation becomes $h_A^{(B)} = \sigma_B^{-1}(\hat{h}_A - \mu_A) + \mu_B$. We now proceed with these features $h_A^{(B)}$ which contain the moments of image $B$ (plane) hidden inside the features of image $A$ (cat). In order to encourage the neural network to pay attention to the injected features of $B$ we modify the loss function to predict the class label $y_A$ and also $y_B$, up to some mixing constant $\lambda \in [0, 1]$. The loss becomes a straight-forward combination

$$\lambda \cdot \ell(h_A^{(B)}, y_A) + (1 - \lambda) \cdot \ell(h_A^{(B)}, y_B).$$

**Implementation.** In practice one needs to apply MoEx only on a single layer in the neural network, as the fused signal is propagated until the end. With PONO as the normalization method, we observe that the first layer ($\ell = 1$) usually leads to the best result. In contrast, we find that MoEx is more suited for later layers when using IN [59], GN [65], or LN [1] for moment extraction. Please see Sec. 5 for a detailed ablation study. The inherent random-ness of mini-batches allows us to implement MoEx very efficiently. For each input instance in the mini-batch $x_i$ we compute the normalized features and moments $\hat{h}_i, \mu_i, \sigma_i$. Subsequently we sample a random permutation $\pi$ and apply MoEx with a random pair within the mini-batch $h_1^{(\pi(i))} \leftarrow F^{-1}(h_1, \mu_{\pi(i)}, \sigma_{\pi(i)})$. See Algorithm 1 in the Appendix for an example implementation in PyTorch [46]. Note that all computations are extremely fast and only introduce negligible overhead during training.

**Hyper-parameters.** To control the intensity of our data augmentation, we perform MoEx during training with some probability $p$. In this way, the model can still see the original features with probability $1 - p$. In practice we found that $p = 0.5$ works well on most datasets except that we set $p = 1$ for ImageNet where we need stronger data augmentation. The interpolation weight $\lambda$ is another hyper-parameter to be tuned. Empirically, we find that 0.9 works well across data sets. The reason can be that the moments contain less information than the normalized features. Please see Appendix for a detailed ablation study.

**Properties.** MoEx is performed entirely at the feature level inside the neural network and can be readily combined with other augmentation methods that operate on the raw input (pixels or words). For instance, Cutmix [72] typically works best when applied on the input pixels directly. We find that the improvements of MoEx are complimentary to such prior work and recommend to use MoEx in combination with established data augmentation methods.

### 4. Experiments

We evaluate the efficacy of MoEx thoroughly across several tasks and data modalities. Our implementation will be released as open source upon publication.

#### 4.1. Image Classification on CIFAR

CIFAR-10 and CIFAR-100 [33] are benchmark datasets containing 50K training and 10K test colored images at
<table>
<thead>
<tr>
<th>Model</th>
<th># of epochs</th>
<th>Test Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>90</td>
<td>23.6</td>
</tr>
<tr>
<td>CutMix [72]</td>
<td>105</td>
<td>23.3</td>
</tr>
<tr>
<td>Mixup [74]</td>
<td>200</td>
<td>22.1</td>
</tr>
<tr>
<td>DropBlock [13]</td>
<td>270</td>
<td>22.4</td>
</tr>
<tr>
<td>Cutout [8]</td>
<td>300</td>
<td>22.2</td>
</tr>
<tr>
<td>Mixup + MoEx</td>
<td>300</td>
<td>21.9</td>
</tr>
<tr>
<td>CutMix + ShakeDrop [70]</td>
<td>300</td>
<td>21.9</td>
</tr>
<tr>
<td>CutMix + ShakeDrop + MoEx</td>
<td>300</td>
<td>21.9</td>
</tr>
</tbody>
</table>

Table 3. Classification results (Test Err (%)) on ImageNet in comparison with various models. Note: The ResNeXt-50 (32×4d) models trained for 300 epochs overfit. They have higher training accuracy but lower test accuracy than the 90-epoch ones.

<table>
<thead>
<tr>
<th>Model</th>
<th># of epochs</th>
<th>Top-1 / Top-5 Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>300</td>
<td>23.1 / 6.8</td>
</tr>
<tr>
<td>CutMix [72]</td>
<td>300</td>
<td>22.2 / 7.7</td>
</tr>
<tr>
<td>Mixup [74]</td>
<td>300</td>
<td>21.9 / 5.9</td>
</tr>
<tr>
<td>DropBlock [13]</td>
<td>300</td>
<td>21.9 / 5.9</td>
</tr>
<tr>
<td>Cutout [8]</td>
<td>300</td>
<td>21.9 / 5.9</td>
</tr>
<tr>
<td>Mixup + MoEx</td>
<td>300</td>
<td>21.9 / 5.9</td>
</tr>
<tr>
<td>CutMix + ShakeDrop [70]</td>
<td>300</td>
<td>21.9 / 5.9</td>
</tr>
<tr>
<td>CutMix + ShakeDrop + MoEx</td>
<td>300</td>
<td>21.9 / 5.9</td>
</tr>
</tbody>
</table>

Table 4. Comparison of regularization and augmentation methods on ImageNet. Stochastic Depth and Cutout results are from [72].

4.2. Image Classification on ImageNet

We evaluate on ImageNet [7] (ILSVRC 2012 version), which consists of 1.3M training images and 50K validation images of various resolutions. For faster convergence, we use NVIDIA’s mixed-precision training code base with batch size 1024, default learning rate 0.1 × batch_size/256, cosine annealing learning rate scheduler [42] with linear warmup [15] for the first 5 epochs. As the model might require more training updates to converge with data augmentation, we apply MoEx to ResNet-50, ResNeXt-50 (32×4d), DenseNet-265 and train them for 90 and 300 epochs. For a fair comparison, we also report Cutmix [72] under the same setting. Since the test server of ImageNet is no longer available to the public, we follow the common practice [13, 22, 69, 72, 74] reporting the validation scores of the baselines as well as our method.

Table 3 shows the test error rates on the ImageNet data set. MoEx is able to improve the classification performance throughout, regardless of model architecture. Similar to the previous CIFAR experiments, we observe in Table 4 that MoEx is highly competitive when compared to existing regularization methods and truly shines when it is combined with them. When applied jointly with CutMix (the strongest alternative), we obtain our lowest Top-1 and Top-5 error of 20.9/5.7 respectively. Beyond, we apply MoEx to EfficientNet-B0 [58] and follow the official train-
ing scheme, we find MoEx is able to help reduce the error rate of baseline from 22.9 to 22.3.

4.3. Fintuneing Imagenet pretrained models on Pascal VOC for Object Detection

To demonstrate that MoEx encourages models to learn better image representations, we apply models pre-trained on ImageNet with MoEx to downstream tasks including object detection on Pascal VOC 2007 dataset. We use the Faster R-CNN [48] with C4 or FPN [40] backbones implemented in Detectron2 [66] and following their default training configurations. We consider three ImageNet pre-trained models: the ResNet-50 provided by He et al. [18], our ResNet-50 baseline trained for 300 epochs, our ResNet-50 trained with CutMix [72], and our ResNet-50 trained with MoEx. A Faster R-CNN is initialized with these pre-trained weights and finetuned on Pascal VOC 2007 + 2012 training data, tested on Pascal VOC 2007 test set, and evaluated with the PASCAL VOC style metric: average precision at IoU 50% which we call AP_{VOC} (or AP50 in detectron2). We also report MS COCO [41] style average precision metric AP_{COCO} which is recently considered as a better choice. Notably, MoEx is not applied during finetuning.

Table 5 shows the average precision of different initializations. We discover that MoEx provides a better initialization than the baseline ResNet-50 and is competitive against CutMix [72] for the downstream cases and leads slightly better performance regardless of backbone architectures.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Initialization</th>
<th>AP_{VOC}</th>
<th>AP_{COCO}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4</td>
<td>ResNet-50 (default)</td>
<td>80.3</td>
<td>51.8</td>
</tr>
<tr>
<td></td>
<td>ResNet-50 (300 epochs)</td>
<td>81.2</td>
<td>53.5</td>
</tr>
<tr>
<td></td>
<td>ResNet-50 + CutMix</td>
<td>82.1</td>
<td>54.3</td>
</tr>
<tr>
<td></td>
<td>ResNet-50 + MoEx</td>
<td>81.6</td>
<td>54.6</td>
</tr>
<tr>
<td>FPN</td>
<td>ResNet-50 (default)</td>
<td>81.8</td>
<td>53.8</td>
</tr>
<tr>
<td></td>
<td>ResNet-50 (300 epochs)</td>
<td>82.0</td>
<td>54.2</td>
</tr>
<tr>
<td></td>
<td>ResNet-50 + CutMix</td>
<td>82.1</td>
<td>54.3</td>
</tr>
<tr>
<td></td>
<td>ResNet-50 + MoEx</td>
<td>82.3</td>
<td>54.3</td>
</tr>
</tbody>
</table>

Table 5. Object detection on PASCAL VOC 2007 test set using Faster R-CNN whose backbone is initialized with different pre-trained models: the ResNet-50 provided by He et al. [18], our ResNet-50 baseline trained for 300 epochs, our ResNet-50 trained with CutMix [72], and our ResNet-50 trained with MoEx. A Faster R-CNN is initialized with these pre-trained weights and finetuned on Pascal VOC 2007 + 2012 training data, tested on Pascal VOC 2007 test set, and evaluated with the PASCAL VOC style metric: average precision at IoU 50% which we call AP_{VOC} (or AP50 in detectron2). We also report MS COCO [41] style average precision metric AP_{COCO} which is recently considered as a better choice. Notably, MoEx is not applied during finetuning.

4.4. 3D model classification on ModelNet

We conduct experiments on Princeton ModelNet10 and ModelNet40 datasets [67] for 3D model classification. This task aims to classify 3D models encoded as 3D point clouds into 10 or 40 categories. As a proof of concept, we use PointNet++ (SSG) [47] implemented efficiently in PyTorch Geometric [11] as the baseline. It does not use surface normal as additional inputs. We apply MoEx to the features after the first set abstraction layer in PointNet++. Following their default setting, all models are trained with ADAM [31] at batch size 32 for 200 epochs. The learning rate is set to 0.001. We tune the hyper-parameters of MoEx on ModelNet-10 and apply the same hyper-parameters to ModelNet-40. We choose $p = 0.5, \lambda = 0.9$, and InstanceNorm for this task, which leads to slightly better results.

Table 6 summarizes the results out of three runs, showing mean error rates with standard errors. MoEx reduces the classification errors from 6.0% to 5.3% and 9.2% to 8.8% on ModelNet10 and ModelNet40, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>ModelNet10 Mean ± Standard Error</th>
<th>ModelNet40 Mean ± Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet++</td>
<td>6.02 ± 0.10</td>
<td>9.16 ± 0.16</td>
</tr>
<tr>
<td>+ MoEx</td>
<td>5.25 ± 0.18</td>
<td>8.78 ± 0.28</td>
</tr>
</tbody>
</table>

Table 6. Classification errors (%) on ModelNet10 and ModelNet40. The mean and standard error out of 3 runs are reported.

5. Ablation Study

5.1. MoEx Design Choices

In the previous section we have established that MoEx yields significant improvements across many tasks and model architectures. In this section we shed light onto which design choices crucially contribute to these improvements. Table 7 shows results on CIFAR-100 with a Resnet-110 architecture, averaged over 3 runs. The column titled MoEx indicates if we performed moment exchange or not.

Label smoothing. First, we investigate if the positive effect of MoEx can be attributed to label smoothing [57]. In label smoothing, one changes the loss of a sample $x$ with label $y$ to $\lambda l(x, y) + \frac{1}{C-1} \sum_{y' \neq y} (1 - \lambda) l(x, y')$, where $C$ denotes the total number of classes. Essentially the neural network is not trained to predict one class with 100% certainty, but instead only up to a confidence $\lambda$.

Further, we evaluate Label Interpolation only. Here, we evaluate MoEx with label interpolation - but without any feature augmentation, essentially investigating the effect of label interpolation alone. Both variations yield some improvements over the baseline, but are significantly worse than MoEx.

Interpolated targets. The last three rows of Table 7 demonstrate the necessity of utilizing the moments for prediction. We investigate two variants: $\lambda = 1$, which corresponds to no label interpolation; MoEx with label smoothing (essentially assigning a small loss to all labels except $y_A$). The last row corresponds to our proposed method, MoEx ($\lambda = 0.9$). We do hyper-parameter search from $p \in \{0.5, 1\}, \lambda \in \{0.5, 0.9\}$ and whether to use PONO or InstanceNorm.
Two general observations can be made: 1) interpolating the labels is crucial for MoEx to be beneficial — the approach leads to absolutely no improvement when we set $\lambda = 1$. 2) it is also important to perform moment exchange, without it MoEx reduces to a version of label smoothing, which yields significantly smaller benefits.

**Choices of normalization methods.** We study how MoEx performs when using moments from LayerNorm (LN) [1], InstanceNorm (IN) [59], PONO [36], GroupNorm (GN) [65], and local response normalization (LRN) [34] perform. For LRN, we use a recent variant [28] which uses the unnormalized 2nd moment at each position. We conduct experiments on CIFAR-100 with ResNet110. For each normalization, we do a hyper-parameter sweep to find the best setup. Table 8 shows classification results of MoEx with various feature normalization methods on CIFAR-100 averaged over 3 runs (with corresponding standard errors). We observe that MoEx generally works with all normalization approaches, however PONO has a slight but significant edge, which we attribute to the fact that it catches the structural information of the feature most effectively. Table 9 shows that different normalization methods work the best at different layers. With PONO or GN, we apply MoEx in the first layer (right before the first stage), whereas the LN moments work best when exchanged before the third stage of a 3-stage ResNet-110; IN is better to be applied right before the second stage. We hypothesize the reason is that PONO moments captures local information while LN and IN compute global features which are better encoded at later stages of a ResNet. For image classification, using PONO seems generally best. While MoEx with other normalization methods in different stages could also obtain competitive results such as LN before Stage 3 in Table 8. We assume it is because in the early layers it is important to exchange the whole mode (and PONO has a significant advantage), whereas for the last stage the scale already contains a lot of information (LN performs best here), which is worth being studied in other architecture such as Transformer [61], etc. Beyond, for some other tasks we observe that using moments from IN can be more favorable (See Subsec. 4.4).

<table>
<thead>
<tr>
<th>Model</th>
<th>Before Stage 1</th>
<th>Before Stage 2</th>
<th>Before Stage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LN</td>
<td>25.9 ± 0.08</td>
<td>25.9 ± 0.07</td>
<td>25.6 ± 0.02</td>
</tr>
<tr>
<td>IN</td>
<td>26.0 ± 0.13</td>
<td>25.7 ± 0.13</td>
<td>26.2 ± 0.13</td>
</tr>
<tr>
<td>GN</td>
<td>25.7 ± 0.09</td>
<td>26.1 ± 0.09</td>
<td>25.8 ± 0.13</td>
</tr>
<tr>
<td>PONO</td>
<td>25.5 ± 0.09</td>
<td>26.1 ± 0.03</td>
<td>26.0 ± 0.09</td>
</tr>
</tbody>
</table>

Table 9. MoEx with different normalization methods applied to different layers in a 3-stage ResNet-110 on CIFAR-100. We bold the best layer of each normalization method.

**5.2. MoEx Hyper-parameters**

In MoEx, $\lambda$ and $1 - \lambda$ serve as the target interpolation weights of labels $y_A$, $y_B$, respectively. To explore the relationship between $\lambda$ and model performance, we train a ResNet50 on ImageNet with $\lambda \in \{0.3, 0.5, 0.7, 0.9\}$ with on PONO. The results are summarized in Table 10. We observe that generally higher $\lambda$ leads to lower error, probably because more information is captured in the normalized features than in the moments. After all, moments only capture general statistics, whereas the features have many channels and can capture texture information in great detail. We also investigate various values of the exchange probability $p$ (for fixed $\lambda = 0.9$), but on the ImageNet data $p = 1$ (i.e. apply MoEx on every image) tends to perform best.

<table>
<thead>
<tr>
<th>Model</th>
<th>$\lambda$</th>
<th>$p$</th>
<th>Top-1 / Top-5 Error(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>1</td>
<td>22.9 / 6.9</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>1</td>
<td>22.2 / 6.4</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>1</td>
<td>21.9 / 6.2</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>1</td>
<td>21.9 / 6.1</td>
<td></td>
</tr>
<tr>
<td>0.95</td>
<td>1</td>
<td>22.5 / 6.3</td>
<td></td>
</tr>
<tr>
<td>0.99</td>
<td>1</td>
<td>22.6 / 6.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 10. Ablation study on ImageNet with different $\lambda$ and $p$ (exchange probability) trained for 300 epochs.

**5.3. Model Analysis**

To estimate the robustness of the models trained with MoEx, we follow the procedure proposed by [20] and

---

<table>
<thead>
<tr>
<th>name</th>
<th>MoEx</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>x</td>
<td>26.3 ± 0.10</td>
</tr>
<tr>
<td>Label smoothing [57]</td>
<td>x</td>
<td>26.0 ± 0.06</td>
</tr>
<tr>
<td>Label Interpolation only</td>
<td>x</td>
<td>26.0 ± 0.12</td>
</tr>
<tr>
<td>MoEx ($\lambda = 1$, not interpolating the labels)</td>
<td>✓</td>
<td>26.3 ± 0.02</td>
</tr>
<tr>
<td>MoEx with label smoothing</td>
<td>✓</td>
<td>25.8 ± 0.09</td>
</tr>
<tr>
<td>MoEx ($\lambda = 0.9$, label interpolation, proposed)</td>
<td>✓</td>
<td>25.5 ± 0.09</td>
</tr>
</tbody>
</table>

Table 7. Ablation study on different design choices.

---

We select the best result from experiments with $\lambda \in \{0.6, 0.7, 0.8, 0.9\}$ and $p \in \{0.25, 0.5, 0.75, 1.0\}$. We choose the best layer among the 1st layer, 1st stage, 2nd stage, and 3rd stage. For each setting, we obtain the mean and standard error out of 3 runs with different random seeds.
evaluate our model on their ImageNet-A data set, which contains 7500 natural images (not originally part of ImageNet) that are misclassified by a publicly released ResNet-50 in torchvision. We compare our models with various publicly released pretrained models including Cutout [74], Mixup [74], CutMix [72], Shape-Resnet [12], and recently proposed AugMix [19]. We report all 5 metrics implemented in the official evaluation code: model accuracy (Acc), root mean square calibration error (RMS), mean absolute distance calibration error (MAD), the area under the response rate accuracy curve (AURRA) and soft F1 [20, 56]. Table 11 summarizes all results. In general MoEx performs fairly well across the board. The combination of MoEx and Cutmix leads to the best performance on most of the metrics.

<table>
<thead>
<tr>
<th>Name</th>
<th>Acc↑</th>
<th>RMS↓</th>
<th>MAD↑</th>
<th>AURRA↑</th>
<th>Soft F1↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50 (torvision)</td>
<td>0</td>
<td>62.6</td>
<td>55.8</td>
<td>0</td>
<td>60.0</td>
</tr>
<tr>
<td>DenseNet-BC-100</td>
<td>2.3</td>
<td>57.8</td>
<td>50.7</td>
<td>1.8</td>
<td>62.1</td>
</tr>
<tr>
<td>VGG-11-BN</td>
<td>3.8</td>
<td>51.1</td>
<td>43.7</td>
<td>3.3</td>
<td>66.8</td>
</tr>
<tr>
<td>WRN-28-10</td>
<td>4.7</td>
<td>54.7</td>
<td>47.8</td>
<td>4.5</td>
<td>62.3</td>
</tr>
<tr>
<td>Mixup</td>
<td>4.4</td>
<td>55.7</td>
<td>48.7</td>
<td>3.8</td>
<td>61.7</td>
</tr>
<tr>
<td>Cutmix</td>
<td>6.6</td>
<td>51.8</td>
<td>44.4</td>
<td>7.0</td>
<td>63.7</td>
</tr>
<tr>
<td>Cutmix + MoEx</td>
<td>7.3</td>
<td>45.0</td>
<td>36.5</td>
<td>7.2</td>
<td>69.3</td>
</tr>
</tbody>
</table>

Table 11. The performance of ResNet-50 variants on ImageNet-A. The up-arrow represents the higher the better, and vice versa.

5.4.4.4. Beyond Computer Vision

We also found that MoEx can be beneficial in other areas such as natural language processing and speech recognition. We use the Speech Command dataset [63] which contains 65000 utterances (one second long) from thousands of people. The goal is to classify them in 30 command words such as "Go", "Stop", etc. There are 56196, 7477, and 6835 examples for training, validation, and test. We use an open source implementation to encode each audio into a mel-spectrogram of size 1x32x32 and feeds it to 2D ConvNets as an one-channel input. We follow the default setup in the codebase training models with initial learning rate 0.01 with ADAM [31] for 70 epochs. The learning rate is reduce on plateau. We use the validation set for hyper-parameter selection and tune MoEx $p \in \{0.25, 0.5, 0.75, 1\}$ and $\lambda \in \{0.5, 0.9\}$. We test the proposed MoEx on three baselines models: DenseNet-BC-100, VGG-11-BN, and WRN-28-10.

Table 12 displays the validation and test errors. We observe that training models with MoEx improve over the baselines significantly in all but one case. The only exception is DenseNet-BC-100, which has only 2% of the parameters of the wide resnet, confirming the findings of Zhang et al. [74] that on this data set data augmentation has little effect on tiny models.

<table>
<thead>
<tr>
<th>Model</th>
<th># Param</th>
<th>Val Err</th>
<th>Test Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenseNet-BC-100</td>
<td>0.8M</td>
<td>3.16</td>
<td>3.23</td>
</tr>
<tr>
<td>+MoEx</td>
<td>0.8M</td>
<td>2.97</td>
<td>3.31</td>
</tr>
<tr>
<td>VGG-11-BN</td>
<td>28.2M</td>
<td>3.05</td>
<td>3.38</td>
</tr>
<tr>
<td>+MoEx</td>
<td>28.2M</td>
<td>2.76</td>
<td>3.00</td>
</tr>
<tr>
<td>WRN-28-10</td>
<td>36.5M</td>
<td>2.42</td>
<td>2.21</td>
</tr>
<tr>
<td>+MoEx</td>
<td>36.5M</td>
<td>2.22</td>
<td>1.98</td>
</tr>
</tbody>
</table>

Table 12. Speech classification on Speech Command. Similar to the observation of [74], regularization methods work better for models with large capacity on this dataset.

In addition, please see Appendix for additional natural language processing results. In contrast to prior augmentation methods, which combine two images in pixel or feature space through linear or non-linear interpolation, MoEx extracts and injects (higher order) statistics about the features. Moments are a natural first choice, but other statistics are possible (e.g. principal components).

6. Conclusion and Future Work

In this paper we propose MoEx, a novel data augmentation algorithm for deep recognition models. Instead of disregarding the moments extracted by the (intra-instance) normalization layer, it forces the neural network to pay special attention towards them. We show empirically that this approach is consistently able to improve classification accuracy and robustness across many data sets, model architectures, and prediction tasks. As an augmentation method in feature space, MoEx is complementary to existing state-of-the-art approaches and can be readily combined with them. Because of its ease of use and extremely simple implementation we hope that MoEx will be useful to many practitioners in computer vision, and beyond — in fact, anybody who trains discriminative deep networks with mini-batches.

Acknowledgments

This research is supported in part by the grants from Facebook, DARPA, the National Science Foundation (IIS-1618134, III-1526012, IIS1149882, IIS-1724282, and TRIPODS-1740822), the Office of Naval Research DOD (N00014-17-1-2175), Bill and Melinda Gates Foundation. We are thankful for generous support by Zillow and SAP America Inc. Facebook has no collaboration with the other sponsors of this project. In particular, we appreciate the valuable discussion with Gao Huang.
References


[57] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In CVPR, 2016. 6, 7


[65] Yuxin Wu and Kaiming He. Group normalization. In ECCV, 2018. 1, 2, 4, 7


[72] 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. 2, 3, 4, 5, 6, 8


[74] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimiza-

A. MoEx PyTorch Implementation

Algorithm 1 shows an example code of MoEx in PyTorch [46].

```python
# x: a batch of features of shape (batch_size, channels, height, width),
# y: onehot labels of shape (batch_size, n_classes)
# norm_type: type of the normalization to use

def moex(x, y, norm_type):
    x, mean, std = normalization(x, norm_type)
    ex_index = torch.randperm(x.shape[0])
    x = x * std[ex_index] + mean[ex_index]
    y_b = y[ex_index]
    return x, y, y_b

# output: model output
# y: original labels
# y_b: labels of moments
# loss_func: loss function used originally
# lam: interpolation weight \(\lambda\)

def interpolate_loss(output, y, y_b, loss_func, lam):
    return lam * loss_func(output, y) + (1. - lam) * loss_func(output, y_b)

def normalization(x, norm_type, epsilon=1e-5):
    # decide how to compute the moments
    if norm_type == 'pono':
        norm_dims = [1]
    elif norm_type == 'instance_norm':
        norm_dims = [2, 3]
    else: # layer norm
        norm_dims = [1, 2, 3]
    # compute the moments
    mean = x.mean(dim=norm_dims, keepdim=True)
    var = x.var(dim=norm_dims, keepdim=True)
    std = (var + epsilon).sqrt()
    # normalize the features, i.e., remove the moments
    x = (x - mean) / std
    return x, mean, std

Algorithm 1. Example code of MoEx in PyTorch.
```

B. MoEx for NLP

B.1. Machine Translation on IWSLT 2014

To show the potential of MoEx on natural language processing (NLP) tasks, we apply MoEx to the state-of-the-art DynamicConv [64] model on 4 tasks in a benchmarking dataset IWSLT 2014 [4]: German to English, English to German, Italian to English, and English to Italian tasks. IWSLT 2014 is based on the transcripts of TED talks and their translation, it contains 167K English and German sentence pairs and 175K English and Italian sentence pairs. We use fairseq library [43] and follow the common setup [10] using 1/23 of the full training set as the validation set for hyper-parameter selection and early stopping. All models are trained with a batch size of 12000 tokens per GPU on 4 GPUs for 20K updates to ensure convergence; however, the models usually don’t improve after 10K updates. We use the validation set to select the best model. We tune the hyper-parameters of MoEx on the validation set of the German to English task including \(p \in \{0.25, 0.5, 0.75, 1.0\}\) and \(\lambda \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}\) and use MoEx with InstanceNorm with \(p = 0.5\) and \(\lambda = 0.8\) after the first encoder layer. We apply the same set of hyper-parameters to the other three language pairs. When computing the moments, the edge paddings are ignored. We use two metrics to evaluate the models: BLEU [44] which is an exact word-matching metric and scaled BERTScore F1 [75].

Table 13 summarizes the average scores (higher better) with standard error rates over three runs. It shows that MoEx consistently improves the baseline model on all four tasks by about 0.2 BLEU and 0.2% BERT-F1. Although these improvements are not exorbitant, they are highly consistent and, as far as we know, MoEx is the first label-perturbing data augmentation method that improves machine translation models.

C. More Examples of MoEx

Figure 4 shows more examples of MoEx. We select top five features out of 64 channels to show here.
Figure 4. MoEx with PONO normalization. The features of image A are normalized and then infused with moments $\mu_B$ (PONO mean), $\sigma_B$ (PONO std) from the image B.