frontiers | Frontiers in Artificial Intelligence

TYPE Original Research
PUBLISHED 04 September 2024
DOI 10.3389/frai.2024.1387936

OPEN ACCESS

EDITED BY
Chunlei Liu,
Children's Medical Research Institute, Australia

REVIEWED BY
Yuqi Han,
Beijing Institute of Technology, China
Hu Wang,
University of Adelaide, Australia
Yufeng Wang,
Beihang University, China

*CORRESPONDENCE
Sarada Krithivasan
✉ sarada.krithi@ibm.com

†PRESENT ADDRESS
Sarada Krithivasan,
IBM Research, Yorktown Heights, NY,
United States

RECEIVED 18 February 2024
ACCEPTED 15 May 2024
PUBLISHED 04 September 2024

CITATION
Krithivasan S, Sen S, Venkataramani S and Raghunathan A (2024) MixTrain: accelerating DNN training via input mixing.
Front. Artif. Intell. 7:1387936.
doi: 10.3389/frai.2024.1387936

MixTrain: accelerating DNN training via input mixing

Sarada Krithivasan[1*†], Sanchari Sen[2], Swagath Venkataramani[2] and Anand Raghunathan[1]

[1]Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States, [2]IBM Research, Yorktown Heights, NY, United States

Training Deep Neural Networks (DNNs) places immense compute requirements on the underlying hardware platforms, expending large amounts of time and energy. An important factor contributing to the long training times is the increasing dataset complexity required to reach state-of-the-art performance in real-world applications. To address this challenge, we explore the use of input mixing, where multiple inputs are combined into a single composite input with an associated composite label for training. The goal is for training on the mixed input to achieve a similar effect as training separately on each the constituent inputs that it represents. This results in a lower number of inputs (or mini-batches) to be processed in each epoch, proportionally reducing training time. We find that naive input mixing leads to a considerable drop in learning performance and model accuracy due to interference between the forward/backward propagation of the mixed inputs. We propose two strategies to address this challenge and realize training speedups from input mixing with minimal impact on accuracy. First, we reduce the impact of inter-input interference by exploiting the spatial separation between the features of the constituent inputs in the network's intermediate representations. We also adaptively vary the mixing ratio of constituent inputs based on their loss in previous epochs. Second, we propose heuristics to automatically identify the subset of the training dataset that is subject to mixing in each epoch. Across ResNets of varying depth, MobileNetV2 and two Vision Transformer networks, we obtain upto 1.6× and 1.8× speedups in training for the ImageNet and Cifar10 datasets, respectively, on an Nvidia RTX 2080Ti GPU, with negligible loss in classification accuracy.

KEYWORDS

deep learning, training, input mixing, runtime efficiency, GPUs (graphics processing units)

1 Introduction

The success of deep neural networks has come at a cost of rapidly rising computational requirements for training. This increase is due to a combination of rising dataset and model complexities. For example, in the context of image classification, training dataset complexity increased significantly from MNIST and CIFAR-10/100 (50,000–60,000 images) to ImageNet-1K (1.2 million) and ImageNet-21K (14.2 million). This is supplemented by a growth in model complexity required to achieve state-of-the-art performance (Stojnic et al., 2023). The impact of increased training computation is both monetary (cost to train) and environmental (CO_2 emissions) (Strubell et al., 2019). A study from OpenAI (Amodei et al., 2018) reports that training costs of deep neural networks have been doubling every 3.5 months, greatly outpacing improvements in hardware capabilities.

1.1 Prior efforts on accelerating DNN training

Several methods have been proposed to accelerate DNN training. We divide them into a few broad categories, such as enabling the use of large-scale parallelism (e.g., hundreds or thousands of servers) in DNN training (Goyal et al., 2017; You et al., 2017), training on reduced-resolution inputs (Touvron et al., 2019; Tan and Le, 2021), training at reduced precision (Sun et al., 2019), pruning to reduce the model size during training (Lym et al., 2019), input instance skipping (Jiang et al., 2019; Zhang et al., 2019) and dataset condensation (Mirzasoleiman et al., 2020; Killamsetty et al., 2021).

1.2 Accelerating DNN training by mixing inputs

Complementary to the aforementioned efforts, we propose the use of input mixing, a technique that has traditionally been used for data augmentation (Zhang et al., 2017; Yun et al., 2019), to accelerate DNN training. Consider two training inputs x_1 and x_2. A mixing function F is applied to x_1 and x_2 to produce a *mixed input* X. The mixed input can be thought of as a point in the input space that combines information from both the constituent inputs that it represents. From the functional perspective, training on a mixed input must produce a similar effect on the model as training on the individual constituent inputs. On the other hand, from a computational viewpoint, mixing inputs reduces the number of input samples that need to be processed during training. This reduction in the effective size of the training dataset leads to fewer mini-batches in each epoch, and thereby lower training time. Due to the nature of input mixing, it is complementary to, and can be combined with, the other approaches to accelerate training described above. In mixTrain, we adopt computationally lightweight mixing operators `CutMix` and `MixUp` that have been proposed for a different purpose, *viz.* data augmentation (Zhang et al., 2017; Yun et al., 2019). As illustrated in Figure 1, `MixUp` performs a simple weighted linear averaging of the pixels of two inputs, while `CutMix` randomly selects a patch of one input and pastes it onto the other.

Realizing training speedups through input mixing raises interesting questions, such as how to train networks on mixed samples, which samples to mix, etc. We observe that indiscriminate application of mixing leads to a considerable drop in learning performance and model accuracy. On further investigation, we find that this can be attributed to the interference between the processing of the constituent inputs within each mixed input. To preserve accuracy, we therefore propose techniques to mitigate this interference. We find that for the `CutMix` operator, the network's internal features largely maintain spatial separation between the constituent inputs in convolutional layers, but this separation is lost in the fully connected layers. We thus propose *split propagation*, wherein the features corresponding to each constituent input are processed separately by the fully connected layers. In contrast, with the `MixUp` operator, spatial separation between the constituent inputs is not maintained. Here, we mitigate the impact of interference through *adaptive mixing*, where the weights of the constituent inputs are varied based on their losses in previous epochs.

Additionally, we explore applying mixing selectively, i.e., only to a subset of training inputs in each epoch. We design a loss-driven metric to identify the training samples that are amenable to mixing in each epoch. We find that inputs at the two ends of the loss distribution, i.e., with very low and very high loss magnitudes, are amenable to mixing. Low-loss inputs are mixed because their functional performance remains largely unaffected by mixing. In contrast, we mix samples with high loss because a considerable percentage of such samples are unlikely to be learned even when no mixing is applied. We show that mixTrain achieves superior accuracy vs. efficiency tradeoffs compared to alternative approaches such as input skipping and early termination. Finally, we note that mixTrain is designed in a completely hyper-parameter free manner. This reduces the additional effort spent on hyper-parameter tuning for different models.

The key contributions of this work can be summarized as follows.

- To the best of our knowledge, mixTrain is the first effort to accelerate DNN training by mixing inputs
- We propose two strategies to improve the learning performance of mixTrain. First, we propose split propagation and adaptive mixing to reduce the impact of interference between the constituent inputs in a composite sample. Second, we apply mixing selectively, i.e., only on a subset of the training dataset in every epoch.
- Across our benchmarks consisting of both image recognition CNNs (including ResNet18/34/50 and MobileNet) and vision transformers, we demonstrate up to 1.6× and 1.8× improvement in training time on the ImageNet and Cifar10 datasets respectively for ∼0.2% Top-1 accuracy loss on a Nvidia RTX 2080Ti GPU, without the use of additional hyper-parameters.

2 Related work

We now discuss related research efforts to accelerate DNN training.

2.1 Hyper-parameter tuning

Many notable efforts are directed toward achieving training efficiency by controlling the hyper-parameters involved in gradient-descent, notably the learning rate and momentum. Akiba et al. (2017), Goyal et al. (2017), and You et al. (2017) propose learning rate tuning algorithms that significantly accelerate training with no loss in accuracy, when distributed to over hundreds of CPU/GPU cores.

2.2 Optimizers with fast convergence

This class of efforts includes optimizers that achieve improved generalization performance within a certain training budget. These

techniques target the evaluation of the weight gradient every iteration- for example, optimizers such as AvaGrad (Savarese et al., 2019) and Adam (Kingma and Ba, 2015) adaptively compute the learning rate across training epochs, resulting in faster convergence than SGD in a similar number of epochs for certain tasks. Similarly, techniques such as (Sutskever et al., 2013) utilize a momentum parameter during training to achieve faster convergence.

2.3 Model size reduction during training

Model size reduction involves dynamically pruning (Yuan et al., 2020; Hoefler et al., 2021) or quantizing (Sun et al., 2019; Fu et al., 2020, 2021; Wolfe and Kyrillidis, 2024) a model during training itself. Training a reduced-capacity model, or with lower-precision results in training speed-ups. In contrast to these techniques which compress the DNN model, MixTrain achieves training speed-up by dynamically reducing the size of the dataset during training.

2.4 Coreset selection strategies

Such techniques select a subset of the training samples that are most informative, i.e., critical to accuracy. These techniques differ in the identification of such critical training samples. Commonly used methods to determine a sample's importance include analyzing sample loss (Jiang et al., 2019; Zhang et al., 2019), gradient-matching techniques (Killamsetty et al., 2021), bi-level optimization methods (Killamsetty et al., 2020), sub-modularity based approaches (Iyer et al., 2020), and decision boundary based methods (Margatina et al., 2021).

3 Input mixing: preliminaries

Input mixing takes multiple inputs and combines them into a composite input, taking in information from each of the constituent inputs. mixTrain uses two operators—`MixUp` (Zhang et al., 2017) and `CutMix` (Yun et al., 2019), which are illustrated in Figure 1.

Consider two inputs, x_1 and x_2. For `MixUp`, as seen in Equation 1, each pixel j of the composite input X is obtained by linearly averaging the corresponding pixels of x_1 and x_2. The mixing ratio r is in the range [0, 1]. The `CutMix` operator selects a random patch of x_1, and pastes it onto x_2. The weightage r of each input x_i is decided by its area in the composite sample.

$$X_j = r \cdot x_{1,j} + (1 - r) \cdot x_{2,j} \tag{1}$$

Further, let us assume the target labels of the constituent inputs are y_1 and y_2. In Zhang et al. (2017) and Yun et al. (2019), the loss of the composite input X is defined as the weighted sum of the loss of X with respect to y_1 and y_2, as shown in Equation 2 for the cross-entropy loss. Here, f is the DNN model, and K the number of classes.

Input mixing has previously been applied for data augmentation, wherein randomly selected training input samples are combined through operators such as (Zhang et al., 2017; Yun et al., 2019) and added to the training set. Training on the randomly combined input samples has the effect of virtually augmenting the dataset, as the model is exposed to new training samples in each epoch. These efforts are focused on improving generalization, often achieved at the cost of increased training time. Specifically, the total number of input samples in each epoch of training after mixing remains the same. Further, in order to realize improvements in accuracy, these techniques often require 2–3× more training epochs than baseline SGD (Zhang et al., 2017; Yun et al., 2019).

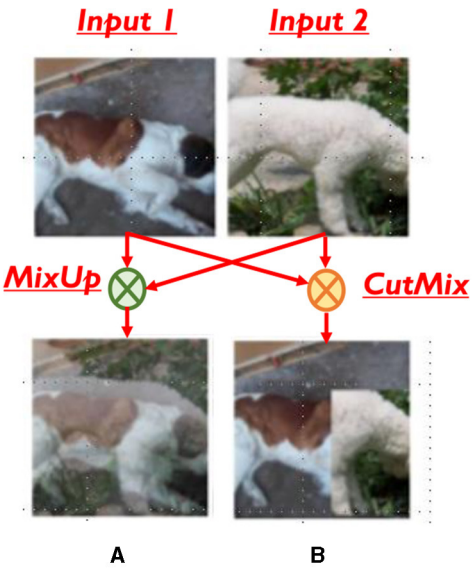

FIGURE 1
Mixing operators **(A)** MixUp **(B)** CutMix. Source: ImageNet.

$$Loss(X) = -(\alpha \cdot log(\frac{e^{f(X)_{y_1}}}{\sum_{l=1}^{K} e^{f(X)_l}}) + (1-\alpha) \cdot log(\frac{e^{f(X)_{y_2}}}{\sum_{l=1}^{K} e^{f(X)_l}})) \tag{2}$$

4 mixTrain: accelerating DNN training via input mixing

The key idea in mixTrain is to improve the overall training time by dynamically applying the mixing operators, `MixUp` and `CutMix`, on the training dataset D to reduce the number of samples in each epoch. However, naive mixing, e.g., where random pairs of input samples are mixed in each training epoch to reduce the number of training samples by half, negatively impacts classification accuracy. As observed in Figure 2A, on the ImageNet-ResNet50 benchmark, the drop in accuracy incurred after training on the reduced (i.e., halved) dataset obtained after applying either operator is nearly 4–6%.

The following subsections discuss the two key strategies that are critical to the overall success of mixTrain, namely, reducing the impact of interference between constituent inputs and selective mixing.

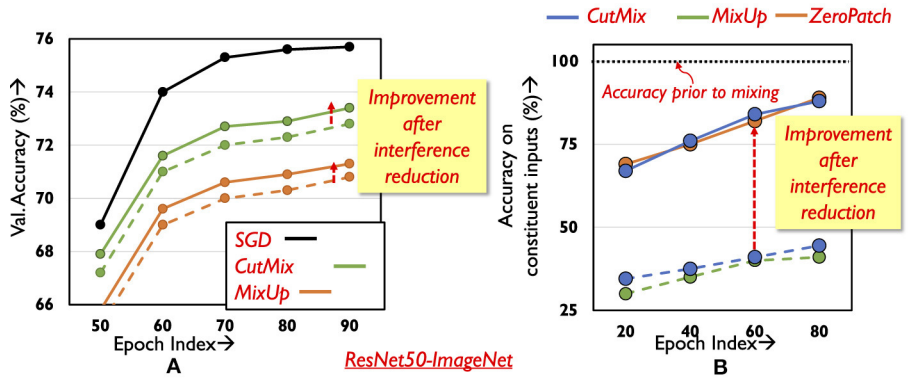

FIGURE 2
Classification performance with mixed inputs. **(A)** Validation accuracy during training. **(B)** Accuracy on constituent inputs for different training methods.

4.1 Reducing impact of interference

In this subsection, we discuss the primary cause affecting the accuracy of training with naive mixing, *i.e.*, interference between constituent inputs, and propose techniques to address the same.

We begin by analyzing the ability of a network trained with mixed inputs to correctly classify the constituent inputs of a composite sample. At different stages of training (different training epochs), we identify the set of training samples that the network classifies correctly without mixing, say set S. Our goal is to understand how the network fares in classifying the samples in set S after they have been mixed. Specifically, we study the network's performance in detecting the presence of both constituent inputs in the mixed sample. Consider inputs x_1 and x_2 in S mixed with ratio $\alpha = 0.5$ to form X, which is passed through the network. The network detects constituent inputs x_1 and x_2 in X, when the softmax scores of their corresponding class labels occupy the highest and second highest positions (order can be inter-changeable between x_1 and x_2). Only a single input is detected when the class label of one of the constituent inputs has the highest softmax score (say x_1), while the second-highest score is achieved by a class not corresponding to the second constituent input (i.e., other than x_2).

Samples in set S are thus mixed in pairs ($r = 0.5$), and the accuracy on the mixed inputs is recorded. Five such runs are conducted to allow for different random input combinations and the results are averaged and presented in Figure 2. Surprisingly, after mixing is applied, the network is able to classify only less than half of the inputs in S (green and blue dotted curves in Figure 2B) even in the final epochs of training- note that these were inputs that were classified correctly without mixing (black line). On further investigation, it is found that for many mixed inputs, the network is able to correctly classify only one of the constituent inputs. The class label of the other constituent input often does not appear even amongst the Top-5 predictions made by the network. This leads to increased loss for one of the constituent samples, consequently impacting training performance and the final validation accuracy. It is thus critical to develop techniques that effectively learn on all constituent samples of a composite input. We next describe our approach to addressing this challenge.

4.1.1 Split propagation

We identify two factors that contribute to the poor classification accuracy of a mixed input's constituent inputs in the case of the `CutMix` operator. Due to the random nature of the patch selected from a constituent input, it is possible to miss the corresponding constituent inputs' class object. Second, there may be interference between the features of the constituent inputs when the network processes the mixed sample. To design effective strategies that improve overall classification performance, it is important to understand the individual effect of each factor. We study the impact of the first factor by passing random patches from the inputs through the network; however, instead of mixing, random patches amounting to half the input area are zeroed-out. As shown using the solid orange curve (ZeroPatch) in Figure 2B, the drop in accuracy is ∼16%, and is significantly lower compared to mixing. This indicates that it is the interference between the constituent inputs that is the primary factor causing degradation in classification performance.

Examining the intermediate representations of the network while processing mixed inputs sheds some light on this interference. By virtue of the nature of convolutions, the spatial separation between constituent inputs in the composite input is maintained through many layers of the network, with only mild interference occurring at the boundaries of the inputs. For example, in Figure 3, the right half of the features in the final convolution layer's output pertain to the right half of the mixed input. The spatial distinction between the features is maintained until the last convolutional layer, but is lost after the averaging action of the final pooling layer. As a result, the fully connected layer correctly classifies only one of the constituent inputs[1].

To aid the network in classifying both constituent inputs correctly, we propose split propagation of constituent features after the final convolution layer. As shown in Figure 3, we identify the region in the final convolutional layer's output maps pertaining to

1 Zhang et al. (2017) and Yun et al. (2019) resolve this issue by exposing the constituent inputs twice in each epoch through two different mixed inputs. While this improves accuracy, it defeats our objective of improving training runtime.

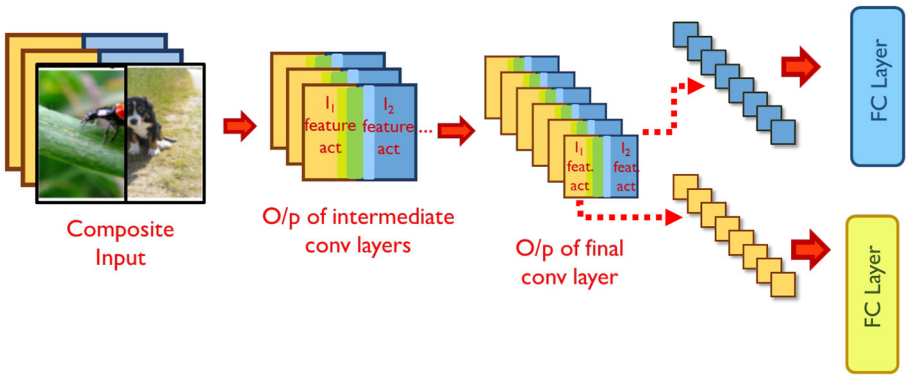

FIGURE 3
Training mixed inputs. Source: ImageNet.

each constituent input, and pass the features separately through the remaining layers of the network. Both constituent inputs of mixed samples are now classified correctly, leading to a significant improvement in classification performance (solid blue curve in Figure 2B). During back-propagation, the output errors of each constituent input are propagated separately until the average pooling layer. The error tensors obtained at the input of the average pooling layer are then concatenated and propagated backwards across the rest of the network. The classification loss for the constituent inputs improves, thereby improving overall validation accuracy (Figure 2A). We note that the split propagation of the constituent inputs can be performed in parallel. Thus, the runtime overheads of this scheme are negligible, accounting for $< 3\%$ of overall training time.

4.1.2 Adaptive mixing

Unlike `CutMix`, the `MixUp` operator averages each element of the constituent inputs prior to feeding them to the network. Therefore, the network's internal representations do not exhibit any spatial separation between the constituent inputs. We thus devise alternative strategies to mitigate the impact of inter-input interference.

It appears from Figure 2A that the validation accuracy with `MixUp` is even lower compared to `CutMix` , due to a slower rate at which training loss improves for the mixed inputs. Naturally, a simple boost in performance can be achieved by at least improving the loss for one of the constituent inputs of the mixed input. We thus adapt the weight (r) of constituent inputs so as to favor the more difficult input, as identified by the loss in the previous epoch. However, if the constituent samples were mixed in the previous epoch, it is not trivial to obtain their individual losses prior to mixing. To that end, we utilize an approximation to evaluate the losses of the constituent inputs in the previous epoch, described as follows. Consider two constituent inputs x_1 and x_2 with target labels y_1 and y_2 respectively, that have been mixed with ratio r_E in epoch E (Equation 3), to form the composite sample X. As seen in Equation 4, we use the loss of the network on the mixed input X to estimate its loss on the individual constituent inputs. Here, K stands for the number of classes in the task. While estimating the loss of x_1 and x_2 in such a manner is indeed an approximation, this allows us to avoid an additional forward propagation step to estimate the true loss of x_1 and x_2, thereby alleviating any runtime overhead.

$$X = r_E * x_1 + (1 - r_E) * x_2 \tag{3}$$

$$Loss(x_1, E) = -log(\frac{e^{f(X)_{y_1}}}{\sum_{l=1}^{K} e^{f(X)_l}}) \quad Loss(x_2, E) = -log(\frac{e^{f(X)_{y_2}}}{\sum_{l=1}^{K} e^{f(X)_l}}) \tag{4}$$

Once the losses of the constituent inputs have been obtained, we mix them in the next epoch $E + 1$ with the ratio r_{E+1} as shown below in Equation 5. As seen in Figure 2A, this provides a boost in classification accuracy.

$$r_{E+1} = \frac{Loss(x_1, E)}{Loss(x_2, E)} \tag{5}$$

Note that there is still some gap between the accuracy with and without mixing even after the use of split propagation and adaptive mixing, which we address next.

4.2 Selective mixing

We explore a second strategy, selective mixing, to further improve accuracy when training with mixed inputs. Here, the general principle is to dynamically identify a subset of the training dataset in each epoch for which mixing does not have a negative impact on overall classification performance. We achieve this through the design of a loss-based metric that determines, for each epoch, the subset of samples S_{mix} that can be mixed in subsequent epochs. Samples that are not amenable to mixing are added to set S_{noMix}. The training dataset is thus formed using samples in S_{noMix}as is, and mixing pairs of samples in S_{mix}.

4.2.1 Overview

The proposed selective mixing strategy consists of three steps as shown in Figure 4. At every epoch, the reduced dataset is divided into mini-batches and fed to the network. The network performs the forward and backward passes on each mini-batch. Once the

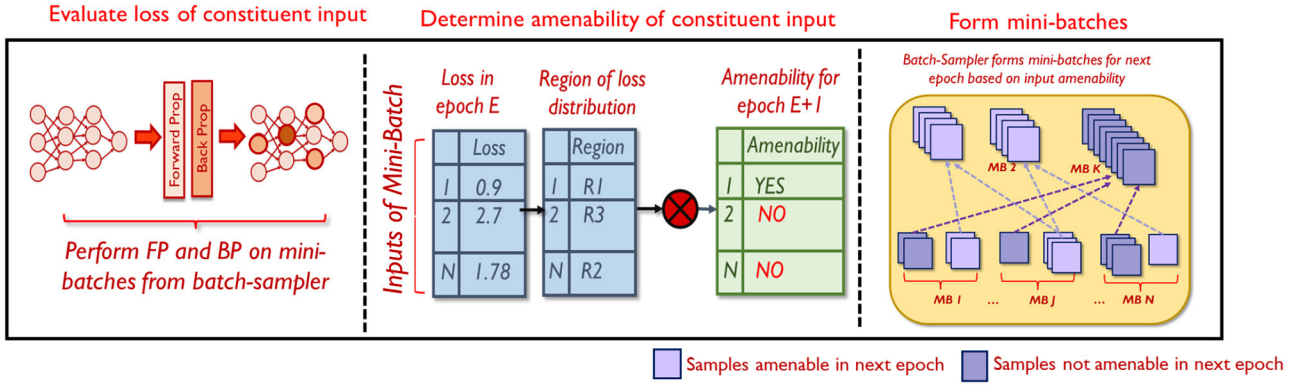

FIGURE 4
Overview of selective mixing.

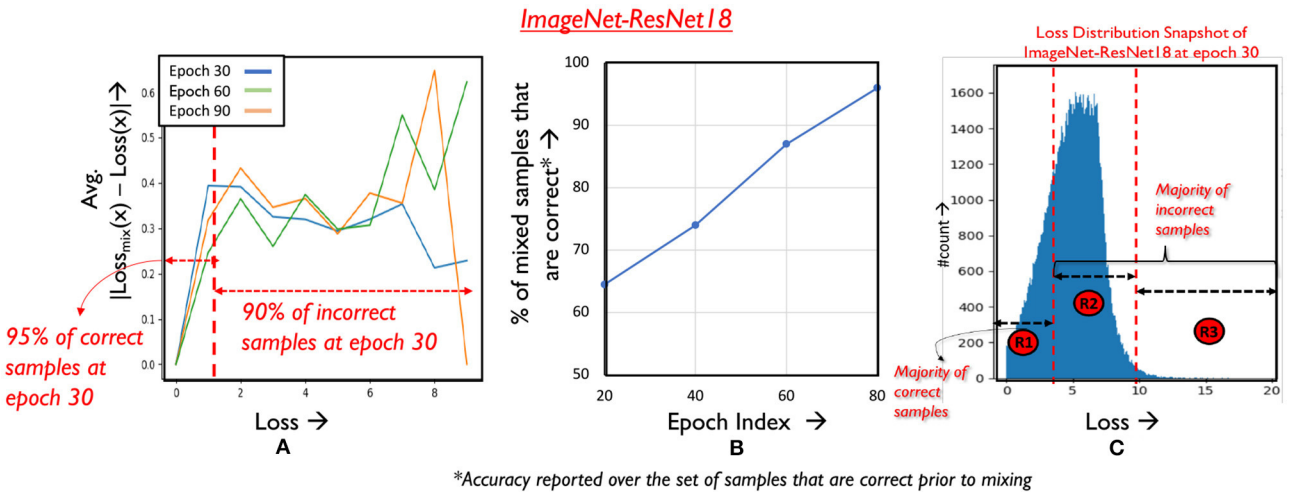

FIGURE 5
Analyzing amenability to mixing. **(A)** Magnitude of difference in loss of a sample before and after mixing. **(B)** Classification performance on mixed samples across epochs. **(C)** Loss distribution of samples in epoch 30.

forward pass for a particular mini-batch is complete, the loss of each constituent input is computed. This is used to determine the amenability of each constituent input to mixing in the next epoch E+1, based on which it is added to S_{mix} or S_{noMix}. Finally, the batch-sampler forms mini-batches for the epoch E+1 by randomly drawing samples from either S_{mix} or S_{noMix}.

The first and the third steps are straight-forward. In the following sub-section, we elaborate on the second step, i.e., determining the amenability of a sample to mixing, in greater detail.

4.2.2 Evaluating amenability to mixing

A suitable loss-based metric must estimate the subsets S_{mix} and S_{noMix} every epoch, such that no negative impact on accuracy is suffered. We design such a metric by studying trends in the loss of a sample prior to and after mixing, at different stages of the training process.

Consider models trained with `MixUp` and `CutMix` at three different training epochs as shown. At each selected epoch, we compute the L_1 difference of the loss of every sample x with and without mixing, i.e., $loss_{mix}(x)$ and $loss(x)$ respectively. We define $loss_{mix}(x)$ as the loss of the mixed sample $x^{'}$ with respect to the golden label of x, as shown in Equation 6. Here, K is the number of classes, and y is the golden label of x. We average $loss_{mix}(x)$ after 5 different random pairings to create $x^{'}$.

$$Loss_{mix}(x) = -log(\frac{e^{f(x^{'})_y}}{\sum_{l=1}^{K} e^{f(x^{'})_l}}) \tag{6}$$

We observe that $loss_{mix}(x)$ deviates and increases further away as $loss(x)$ increases, consistently across the benchmarks analyzed for both operators (Figure 5A depicts the same for `CutMix`). In other words, the graph indicates that *as loss(x) increases, its amenability to mixing decreases*. Furthermore, we find that prior to mixing, a majority of the correctly classified samples occupy the low loss regime as shown in Figure 5A. After applying mixing to these

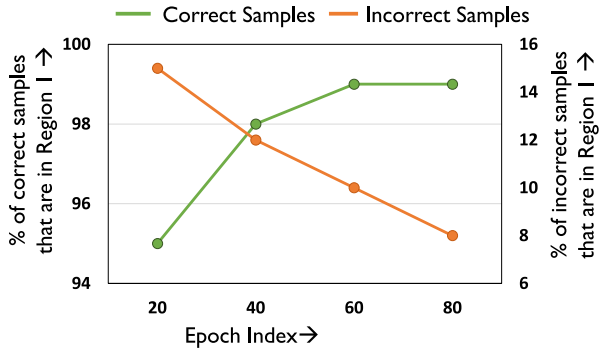

FIGURE 6
Efficacy of threshold L_{mid}.

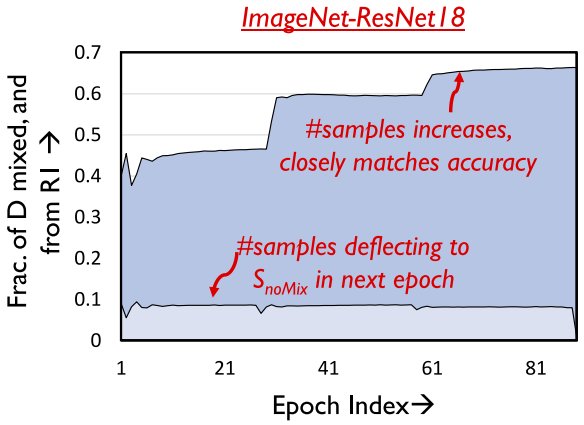

FIGURE 7
Amenability of Region 1.

samples, we find that their classification accuracy is largely retained, especially as epochs progress, as depicted in Figure 5B for the `CutMix` operator.

Hence, for samples that are not mixed in epoch E, we determine their amenability to mixing in the next epoch based on the particular region of the loss distribution it belongs to. As illustrated in Figure 5C, the loss distribution is divided into three regions that utilize different criteria for gauging amenability. We now discuss the criteria for each region, and the conditions for continuing mixing in subsequent epochs.

Region 1 corresponds to the area in the loss distribution where a majority of the correctly classified samples are located. From Figure 5B we know that the loss, and to a certain extent the classification accuracy of such samples remains largely unaffected by mixing and such samples are hence mixed aggressively. Next, we consider the portion of the loss distribution occupied by the incorrect samples and divide this space into two regions. Region 2 comprises of incorrect samples with moderate loss. To avoid any negative impact on accuracy, we avoid mixing these samples. Moving on to Region 3, these are samples the network finds very difficult to classify as characterized by their high loss magnitudes. We find that the training effort can be reduced on samples that consistently occur in Region 3 by mixing them, as they are unlikely to contribute to final classification accuracy.

The separations in the loss distribution are realized using simple linear clustering techniques that correlate the loss of a training sample in some epoch E to classification accuracy, based on trends in previous epochs. Let L_{corr} and L_{incorr} represent the running average of the correct and incorrect samples in S_{noMix} respectively (calculated from epoch 0 to E-1), and let L_{mid} denote the average of the two quantities as shown in Equation 7, i.e.,

$$L_{mid} = 0.5 * (L_{corr} + L_{incorr}) \tag{7}$$

L_{mid} acts as a boundary between the correct and incorrect samples, effectively creating two clusters whose centroids are given by L_{corr} and L_{incorr}. Thus, samples with loss less than L_{mid} in epoch E can be identified as Region 1 samples, as they are likely to be correct. Figure 6 plots the efficacy of L_{mid} across different epochs (fraction of correct inputs under L_{mid}). As desired, a majority of the correct samples ($> 95\%$) fall in Region 1, while only including a negligible fraction of incorrect samples ($< 10\%$). Furthermore, samples with loss greater than L_{incorr} in a particular epoch are in the upper percentile of the loss distribution of the incorrect samples. L_{incorr} can hence used to create Region 2 and Region 3 as shown. We note that loss thresholds of better quality can potentially be identified by introducing hyper-parameters. However, tuning these hyper-parameters for each network separately is a costly process, diminishing the runtime benefits achieved by reducing training complexity.

We will now discuss the amenability criteria designed for samples belonging to Regions 1 and 3.

4.2.3 Amenability criteria for Region 1

Consider a sample A belonging to Region 1 in epoch E, i.e., $Loss_A < L_{mid}$. From Figure 5B it is known that samples in Region 1 are likely to be correctly classified prior to mixing. We mix such samples as long as their loss does not exceed L_{mid} at some later epoch E', i.e., likely to be classified incorrectly. After epoch E', they are shifted to S_{noMix}. Figure 7 illustrates the temporal variation in the number of samples that are in S_{mix}, and from Region 1 of the loss distribution. As can be seen, the number of such samples increases across epochs. This is because as epochs progress classification accuracy improves, thereby resulting in more samples having loss below L_{mid}, i.e., belonging to Region 1. We note that using a loss-based threshold to determine amenability to mixing is more robust instead of directly using classification performance [Section 1 (Appendix)], as we find that mixing outlier samples, i.e., samples with high loss yet correct classification affects overall accuracy.

The graph also depicts the fraction of samples that move to S_{noMix} every epoch, which is a very small fraction of the samples that are mixed. This justifies the design of the amenability rule for Region 1.

4.2.4 Amenability criteria for Region 3

Samples in Region 3 have high loss ($loss > L_{incorr}$), and are generally very difficult to classify by the network even if

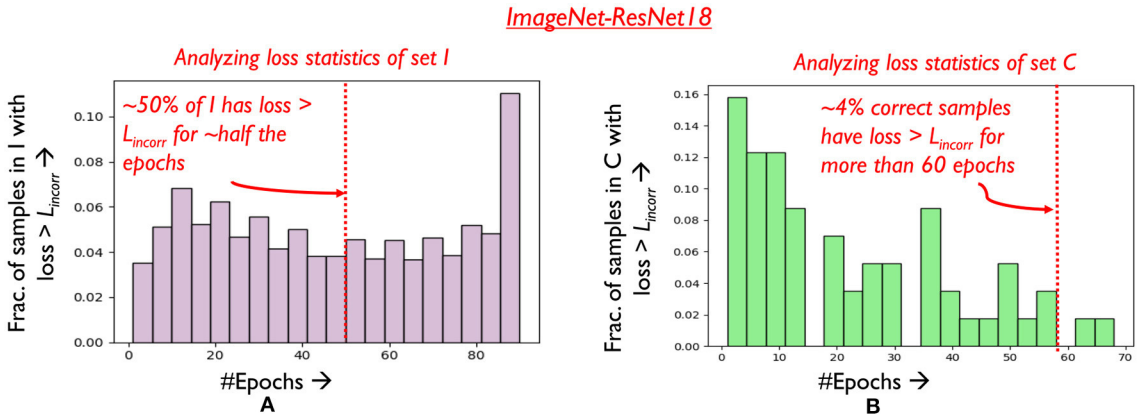

FIGURE 8
Analyzing **(A)** loss distribution of samples in set I **(B)** loss distribution of samples in set C.

they are trained without mixing. In fact, we observe that a considerable fraction of samples that consistently occur in Region 3 across epochs remain incorrect at the end of the training process. Let I denote the set of such samples that are incorrect when training concludes. We plot a histogram of the number of epochs samples in I occupy Region 3 across training in Figure 8A. Clearly, it is observed that over half the samples in I consistently occur in Region 3 for over 70% of the training process. It can thus be argued from a practical runtime efficiency perspective that training effort on such samples can be reduced using mixing. Some challenges however persist. As classification statistics evolve during training, it is difficult to determine which samples to mix at earlier epochs, without negatively affecting final classification accuracy. Consider set C, which comprises of samples that are correctly classified at the end of training. In Figure 8B, it is seen that around 4% of the samples in C occur in Region 3 for over 60% of the training process, with their classification accuracy improving only in the later stages of training. We must thus stipulate criteria to identify the desired subset of Region 3 samples that can be mixed.

To that end, we target samples that the network finds difficult to classify in the current epoch. In addition to belonging to Region 3, if a sample's loss increases over consecutive epochs (i.e., become increasingly difficult) it is mixed for the next epoch, following which it is brought back to S_{noMix}. In Figure 9B, we find that increasing the period of time k for which the difficult samples must exhibit increasing loss and subsequently be mixed, only marginally improves the accuracy and runtime benefits. We hence use $k = 1$ for all our experiments thereby eliminating our dependence on any hyper-parameters. The temporal variation in the fraction of Region 3 samples mixed every epoch is depicted in Figure 9A. This fraction decreases across epochs, since several samples in Region 3 shift to Region 1 as accuracy improves. Interestingly, mixing difficult samples provides $\sim$ 0.2% boost in classification performance over the overall validation set across all our benchmarks, as opposed to training them without mixing. We believe this has the effect of allowing the network to focus on samples with moderate loss, that are more likely to contribute to final accuracy. Finally, we highlight the advantage of mixing such difficult samples instead of skipping them in Section 5.

Determining sample amenability in each epoch adds not more than 2% overhead in runtime on average, and 4% additional storage costs. The proposed amenability criteria thus help us successfully realize selective mixing, i.e., achieve a competitive runtime efficiency vs. accuracy trade-off.

5 Experimental results

We showcase the runtime benefits achieved by mixTrain across different classes of image recognition DNNs, namely convolutional neural networks (i.e., CNNs) and vision transformers (Dosovitskiy et al., 2020). We consider two datasets, namely ImageNet (Deng et al., 2009) and Cifar10 (Krizhevsky et al., 2010). The benchmarks for the ImageNet dataset consist of four image-recognition CNNs, *viz.* ResNet18, ResNet34, ResNet50 (He et al., 2015) and MobileNetV2 (Sandler et al., 2018), trained using the same training hyper-parameters such as learning rate, epochs etc., as in He et al. (2015) and Sandler et al. (2018). With regards to the Cifar10 dataset, we consider the ResNet18 and Resnet34 image-recognition CNNs (He et al., 2015) (see Appendix for results). We also consider three vision transformer architectures, ViT-small, ViT-SWIN and ViT-pretrained. Details on the vision transformer architectures, and training hyper-parameters for all benchmarks can be found in Section 1.1 (Appendix).

Across all benchmarks, we report the speed-up achieved by mixTrain over the same number of epochs as the baseline, by comparing wall-clock times.

5.1 Execution time benefits

5.1.1 ImageNet

Table 1 presents the training performance of baseline SGD and mixTrain on different ImageNet benchmarks in terms of the Top-1 classification error and speed-up. On average, across all benchmarks, mixTrain mixes nearly 48% and 68% of the

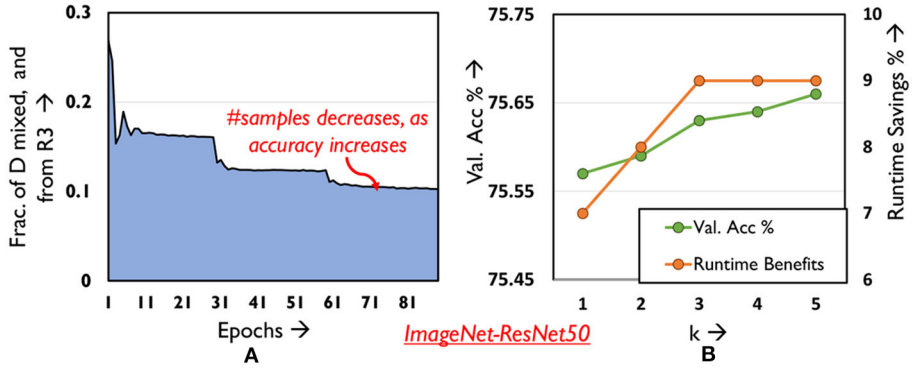

FIGURE 9
Analyzing **(A)** distribution of mixed samples from R3 across epochs **(B)** validation accuracy and runtime savings as hyper-parameter *k* is varied.

TABLE 1 Training CNNs on ImageNet.

Network	Training strategy	Top-1 error	Speed-up
ResNet18	Baseline SGD	30.2%	1×
	mixTrain-**CutMix**	**30.44%**	**1.51×**
	mixTrain-**MixUp**	**30.6%**	**1.32×**
ResNet34	Baseline SGD	26%	1×
	mixTrain-**CutMix**	**26.25%**	**1.54×**
	mixTrain-**MixUp**	**26.4%**	**1.37×**
ResNet50	Baseline SGD	24.3%	1×
	mixTrain-**CutMix**	**24.45%**	**1.56×**
	mixTrain-**MixUp**	**24.6%**	**1.41×**
MobileNetV2	Baseline SGD	28.5%	1×
	mixTrain-**CutMix**	**28.76%**	**1.52×**
	mixTrain-**MixUp**	**29%**	**1.3×**

Bold values represent results using proposed approach.

TABLE 2 Training vision transformers on Cifar10.

Network	Training strategy	Top-1 error	Speed-up
ViT-small (Training from scratch)	Baseline SGD	19%	1×
	mixTrain-**MixUp**	**19.11%**	**1.37×**
	mixTrain-**CutMix**	**19.35%**	**1.32×**
ViT-SWIN (Training from scratch)	Baseline SGD	9%	1×
	mixTrain-**MixUp**	**8.9%**	**1.44×**
	mixTrain-**CutMix**	**9.2%**	**1.4×**
ViT-pretrained (Fine-tuning)	Baseline SGD	2.5%	1×
	mixTrain-**MixUp**	**2.46%**	**1.6×**
	mixTrain-**CutMix**	**2.55%**	**1.58×**

Bold values represent results using proposed approach.

training dataset per epoch with `MixUp` and `CutMix` respectively. As can be seen, `CutMix` achieves a slightly superior trade-off than `MixUp` across all benchmarks, achieving upto around 1.6× reduction in runtime compared to to the baseline, while sacrificing only ∼0.2% loss in Top-1 accuracy. This is primarily because interference between constituent samples is better mitigated through split propagation, thereby resulting in more inputs being mixed.

5.1.2 Cifar10

We present our runtime and accuracy trade-off achieved on the Cifar10 vision transformer benchmarks in Table 2. As can be seen, mixTrain achieves 1.3×-1.6× training speed-up for nearly no loss in accuracy. This clearly underscores that mixTrain is directly applicable to any image classification DNN, regardless of the architecture or backbone deployed. Further, our results in Table 2 also indicate that mixTrain is not only applicable to training vision transformers from scratch, but to the fine-tuning stage as well. In Section 1.2 (Appendix) we discuss the speed-ups achieved by mixTrain on the CNN benchmarks trained on Cifar10.

5.1.3 Runtime overhead analysis

Across all our benchmarks, we observe that mixTrain adds no more than 2% overhead in runtime. These marginal overheads arise due to (i) calculating amenability of inputs to mixing and (ii) split propagation (for Cut-Mix). In (i) we compare the sample's loss against some thresholds, and update thresholds every epoch. However, these simple scalar operations have negligible runtime (<1.5% overhead) compared to the multiple GEMM operations performed during training. For (ii), during split propagation, the FC layers process the constituent inputs separately. However, the FC layers now operate on inputs of smaller size (i.e., corresponding to the size occupied by the features of the constituent input, which is nearly half the size of the original input). Thus, split propagation also adds less than <1% runtime overhead compared to the baseline.

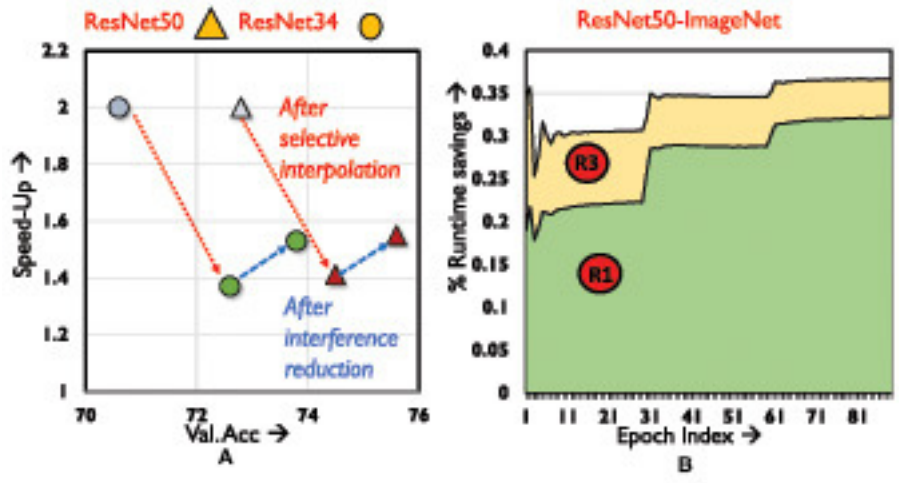

FIGURE 10
Ablation analysis. **(A)** Training speed-up vs. validation accuracy. **(B)** Variation in runtime savings across training epochs.

5.2 Ablation

In this subsection we conduct an ablation analysis of mixTrain.

5.2.1 Contribution of interference reduction and selective mixing

mixTrain uses two strategies to achieve an optimal accuracy vs. runtime trade-off, i.e., reducing impact of interference and selective mixing. Figure 10A depicts the contribution of each strategy toward runtime savings, for the `CutMix` operator.

The light blue markings indicate naive mixing. Selective mixing automatically identifies a subset of training samples that can be mixed every epoch such that classification accuracy is not impacted. However, if interference between the constituent inputs is not mitigated, training performance on mixed samples is poor (green markings). Consequently, the selective mixing strategy is forced to become conservative, identifying fewer samples that can be mixed every epoch without affecting accuracy severely. Reducing interference between the constituent inputs improves accuracy by more than 1%, and speed-up by 10% (red markings).

5.2.2 Breakdown of selective mixing

We breakdown selective mixing by examining the region of the loss distribution that provides the most benefits. From Figure 10B (generated using `CutMix`) it is evident that Region 1 samples provide the bulk of our benefits on the ResNet18-ImageNet benchmark, accounting for nearly 25% of the savings. This is because as training progresses, a majority of training samples fall in Region 1 (i.e., become easier to classify). Interpolating Region 3 samples, accounts for additional 8% runtime savings.

5.3 Quantitative comparison study

We compare the performance of mixTrain against competing methods that accelerate DNN training.

5.3.1 Sample skipping

As a representative of sample skipping, we specifically consider the performance of Zhang et al. (2019) (Figure 11A) and Jiang et al. (2019) (Figure 11B) on the ResNet50 benchmark. In these techniques, samples that the network finds easy to classify, as identified by low classification loss, are skipped thereby resulting in fewer mini-batches as training proceeds. Two issues are typically encountered by such techniques. First, as no training is conducted on the samples that are skipped, this subset is often a small, conservative fraction of the training dataset. Second, additional overhead is incurred in each epoch to determine this subset, as it is non-trivial to estimate the most recent loss of samples that had been discarded in previous epochs. In Figure 11B, we implement (Jiang et al., 2019) and overlook the overheads associated in determining the subset of samples that must be skipped, and report the resulting runtime across epochs.

Clearly, mixTrain achieves better model accuracy and runtime benefits against both efforts, even when overheads are overlooked. As the network is ultimately trained on every input in each epoch, we reduce the number of minibatches more aggressively, while incurring negligible overheads incurred to form S_{mix} and S_{noMix}. Finally, we analyze (Figure 11C) the accuracy if Region3 samples were to be skipped instead of mixed, using the same policy discussed in Section 4.2 for different values of k. Clearly, mixTrain achieves better convergence, allowing it to leverage runtime benefits from this region.

5.3.2 Coreset selection techniques

In the table below, we compare the performance of MixTrain-CutMix against three popular coreset selection techniques: Glister (Killamsetty et al., 2020), Grand (Paul et al., 2021) and Facility-location based methods (Iyer et al., 2020). Similar to mixTrain, coreset selection techniques aim to reduce training runtime by reducing the number of mini-batches to train every epoch, by identifying a subset of training data-points that are critical to accuracy. Such techniques perform better than random sampling (i.e., better accuracy), when the fraction of the training

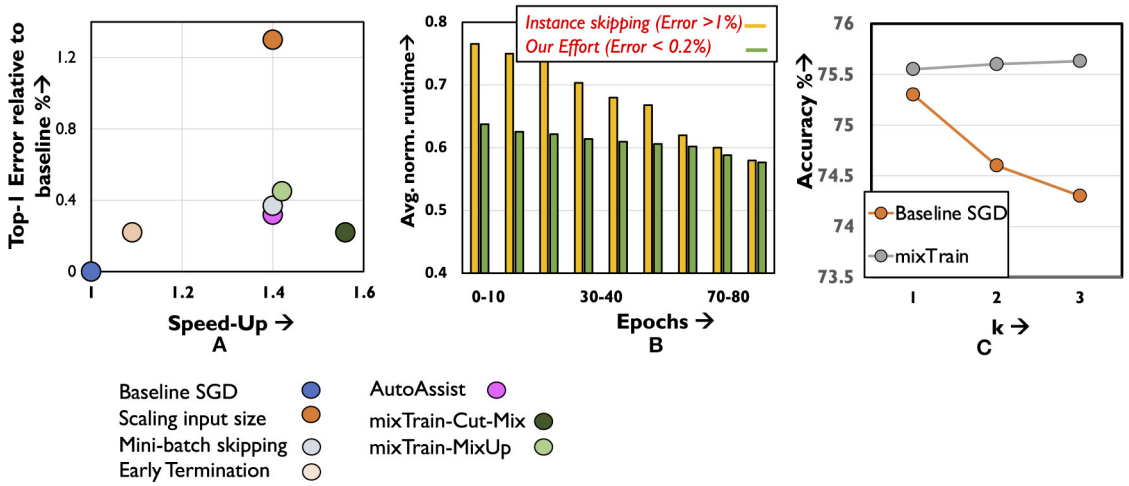

FIGURE 11
Results on **(A)** quantitative comparison of mixTrain against existing training acceleration techniques. **(B)** Comparison against instance skipping. **(C)** Ablation analysis for studying impact of hyper-parameter *k* on accuracy.

TABLE 3 Comparison against coreset selection techniques.

Training method	Average fraction of the dataset used for training across epochs	Top-1 error	Speed-up
Baseline	1	4.4%	1×
mixTrain-MixUp	**0.69**	**4.33%**	**1.4×**
mixTrain-CutMix	**0.66**	**4.2%**	**1.45×**
Glister	0.8	4.65	1.18×
	0.7	4.76%	1.32×
Grand	0.8	4.6%	1.15×
	0.7	4.7%	1.2×
Facility location	0.8	4.55%	1.19×
	0.7	4.79%	1.25×

Bold values represent results using proposed approach.

dataset retained is low (Guo et al., 2022). However, as can be seen in Table 3, these techniques require a large fraction of the training dataset in order to remain iso-accurate with the baseline. mixTrain clearly achieves a better accuracy vs. speed-up trade-off.

5.3.3 Other approximations

We consider three approximation strategies, i.e., early termination, mini-batch skipping and input size scaling (Figure 11A). For early-termination, we stop baseline SGD training at an earlier epoch when it achieves the same accuracy as mixTrain, and report the resulting runtime benefits. Next, for mini-batch skipping we stochastically skip *s*% of the mini-batches every epoch, and for input size scaling, we train on inputs scaled down by some factor *s*. For the Imagenet benchmark highlighted in Figure 11A, 30% of the mini-batches were skipped randomly every epoch. Likewise, a scaling factor of $s = 1.4\times$ was used for input size scaling. In both cases, the parameter *s* is selected such that it is iso-runtime with mixTrain. Clearly, in all three cases, mixTrain achieves a superior accuracy vs. runtime trade-off as seen for the ResNet50 benchmark.

6 Conclusion

We introduce a new approach to improve the training efficiency of state-of-the-art DNNs by utilizing input mixing. We propose mixTrain that comprises of two strategies to achieve an acceptable accuracy vs. speed-up trade-off. First, we propose split propagation and adaptive mixing to reduce the impact of interference between the constituent inputs in a composite sample. Second, we apply mixing selectively, i.e., only on a subset of the training dataset every epoch. Across DNNs on the ImageNet dataset, we achieve upto a 1.6× improvement in runtime for ∼0.2% loss in accuracy.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

SK: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. SS: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. SV:

Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. AR: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This work was supported in part by Semiconductor Research Corporation (SRC).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frai.2024.1387936/full#supplementary-material

References

Akiba, T., Suzuki, S., and Fukuda, K. (2017). "Extremely large minibatch SGD: training resnet-50 on imagenet in 15 minutes," in *CoRR, abs/1711.04325*.

Amodei, D., Hernandez, D., Sastry, G., Clark, J., Brockman, G., and Sutskever, I. (2018). *Deep Neural Network Training Costs*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "ImageNet: a large-scale hierarchical image database," in CVPR09.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2020). "An image is worth 16x16 words: transformers for image recognition at scale," in *CoRR, abs/2010.11929*.

Fu, Y., Guo, H., Li, M., Yang, X., Ding, Y., Chandra, V., et al. (2021). "CPT: efficient deep neural network training via cyclic precision," in *CoRR, abs/2101.09868* (Vancouver, BC: Curran Associates Inc.).

Fu, Y., You, H., Zhao, Y., Wang, Y., Li, C., Gopalakrishnan, K., et al. (2020). "Fractrain: Fractionally squeezing bit savings both temporally and spatially for efficient dnn training," in *Advances in Neural Information Processing Systems*, 33.

Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., et al. (2017). "Accurate, large minibatch SGD: training imagenet in 1 hour," in *CoRR, abs/1706.02677*.

Guo, C., Zhao, B., and Bai, Y. (2022). "Deepcore: a comprehensive library forcoreset selection indeep learning," in *Database and Expert Systems Applications*, eds. C. Strauss, A. Cuzzocrea, G. Kotsis, A. M. Tjoa, and I. Khalil (Cham. Springer International Publishing), 181–195.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). "Deep residual learning for image recognition," in *CoRR, abs/1512.03385*.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. (2021). Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.* 22, 1–124. doi: 10.5555/3546258.3546499

Iyer, R. K., Khargoankar, N., Bilmes, J. A., and Asanani, H. (2020). "Submodular combinatorial information measures with applications in machine learning," in *CoRR, abs/2006.15412*.

Jiang, A. H., Wong, D. L., Zhou, G., Andersen, D. G., Dean, J., Ganger, G. R., et al. (2019). "Accelerating deep learning by focusing on the biggest losers," in *CoRR, abs/1910.00762*.

Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., De, A., and Iyer, R. K. (2021). "GRAD-MATCH: gradient matching based data subset selection for efficient deep model training," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, eds. M. Meila, and T. Zhang (New York: PMLR), 5464–5474.

Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., and Iyer, R. K. (2020). "GLISTER: generalization based data subset selection for efficient and robust learning," in *CoRR, abs/2012.10630*.

Kingma, D. P., and Ba, J. (2015). "Adam: a method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015*, Y. Bengio, and Y. LeCun (San Diego, CA: Conference Track Proceedings).

Krizhevsky, A., Nair, V., and Hinton, G. (2010). *Cifar-10 (Canadian Institute for Advanced Research)*.

Lym, S., Choukse, E., Zangeneh, S., Wen, W., Erez, M., and Shanghavi, S. (2019). "Prunetrain: Gradual structured pruning from scratch for faster neural network training," in *CoRR, abs/1901.09290* (New York, NY: Association for Computing Machinery).

Margatina, K., Vernikos, G., Barrault, L., and Aletras, N. (2021). "Active learning by acquiring contrastive examples," in *CoRR, abs/2109.03764*.

Mirzasoleiman, B., Bilmes, J., and Leskovec, J. (2020). "Coresets for data-efficient training of machine learning models," in *Proceedings of the 37th International Conference on Machine Learning*, ed. Singh (New York: PMLR), 6950–6960.

Paul, M., Ganguli, S., and Dziugaite, G. K. (2021). "Deep learning on a data diet: finding important examples early in training," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan (New York: Curran Associates, Inc), 20596–20607.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. (2018). "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," in *CoRR, abs/1801.04381*.

Savarese, P., McAllester, D., Babu, S., and Maire, M. (2019). "Domain-independent dominance of adaptive methods," in *CoRR, abs/1912.01823* (Los Alamitos, CA: IEEE Computer Society).

Stojnic, R., Taylor, R., and Kardas, M. (2023). *Imagenet Leaderboard*.

Strubell, E., Ganesh, A., and McCallum, A. (2019). *Energy and Policy Considerations for Deep Learning in NLP*.

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V., et al. (2019). "Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks," in *NeurIPS* (Red Hook, NY: Curran Associates Inc.).

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, eds. S. Dasgupta, and D. McAllester (Atlanta: PMLR).

Tan, M., and Le, Q. V. (2021). "Efficientnetv2: Smaller models and faster training," in *CoRR, abs/2104.00298*.

Touvron, H., Vedaldi, A., Douze, M., and Jégou, H. (2019). "Fixing the train-test resolution discrepancy," in *CoRR, abs/1906.06423* (Curran Associates, Inc.).

Wolfe, C. R., and Kyrillidis, A. (2024). Better schedules for low precision training of deep neural networks. *Mach Learn*. 113, 3569–3587. doi: 10.1007/s10994-023-06480-0

You, Y., Gitman, I., and Ginsburg, B. (2017). "Scaling SGD batch size to 32k for imagenet training," in *CoRR, abs/1708.03888*.

Yuan, X., Savarese, P., and Maire, M. (2020). "Growing efficient deep networks by structured continuous sparsification," in *CoRR, abs/2007.15353*.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *CoRR, abs/1905.04899*.

Zhang, H., Cissé, M., Dauphin, Y. N., and Lopez-Paz, D. (2017). "mixup: Beyond empirical risk minimization," in *CoRR, abs/1710.09412*.

Zhang, J., Yu, H., and Dhillon, I. S. (2019). "Autoassist: A framework to accelerate training of deep neural networks," in *CoRR, abs/1905.03381* (Curran Associates, Inc.).