InterTrain: ACCELERATING DNN TRAINING VIA INPUT INTERPOLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

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 this end, we propose to reduce training runtimes by combining a subset of inputs in the training dataset via an interpolation operation. The goal is for training on the interpolated 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. However, we find that naively interpolating inputs leads to a considerable drop in learning performance and model accuracy. This is because the efficacy of learning on interpolated inputs is reduced by the interference between the forward/backward propagation of their constituent inputs. We propose two strategies to address this challenge and realize training speedups with minimal impact on accuracy. First, we reduce the impact of interference by exploiting the spatial separation between the features of the constituent inputs in the network's intermediate representations. We also adaptively vary the weightage of constituent inputs based on their loss in previous epochs. Second, we propose loss-based metrics to automatically identify the subset of the training dataset that is subject to interpolation in each epoch. For ResNets of varying depth and MobileNetV2, 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.

1 INTRODUCTION

The success of deep neural networks has led to their widespread application in several domains of machine learning involving images (Krizhevsky et al., 2017), videos (Ng et al., 2015), text (Zhou et al., 2015) and natural language (Goldberg & Hirst, 2017). However, this success has come at the cost of rising dataset and model complexities. For example, the ImageNet-1K dataset (Deng et al., 2009) contains over 1.2 million training images, while the larger ImageNet-21K dataset contains around 14.2 million images. This is supplemented by a growth in model complexity (He et al., 2015; Tan & Le, 2021) required to achieve improved performance. Together, these factors contribute to the rapid increase in the computational requirements of DNN training, with the impact being felt both monetarily (cost to train) and environmentally (CO_2 emissions) (Strubell et al., 2019). A study from OpenAI (Amodei et al., 2018) reports that training costs have been doubling every 3.5 months, easily outpacing improvements in hardware capabilities.

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 improved optimization algorithms (Kingma & Ba, 2015; Zhang & Mitliagkas, 2018), enabling the use of large-scale parallelism (e.g., hundreds or thousands of servers) in DNN training (You et al., 2017a; Goyal et al., 2017; You et al., 2017b), training on reduced-resolution inputs (Touvron et al., 2020; Tan & Le, 2021), training at reduced precision (Sun et al., 2019), pruning to reduce the model size during training (Lym et al., 2019; Huang et al., 2016), and instance skipping strategies (Jiang et al., 2019; Zhang et al., 2019).

InterTrain**: Accelerating DNN Training with Input Interpolation**: Complementary to previous efforts, we propose the use of *input interpolation to accelerate DNN training*. Consider training inputs x_1, .. x_N. An interpolation function F, is applied to x_1, .. x_N to produce a *interpolated input* X. The interpolated input can be thought of as a point in the input space that combines information from all of the constituent inputs that it represents. From the functional perspective,

training on an interpolated input must produce a similar effect on the model as training on the individual constituent inputs. On the other hand, from a computational viewpoint, interpolation 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 each epoch, thereby resulting in lower training time. Due to the nature of input interpolation, it is complementary to, and can be combined with, the other approaches to accelerate training described above. In InterTrain, we adopt computationally lightweight interpolation operators that have been utilized for a different purpose, *viz.* data augmentation (Zhang et al., 2017; Yun et al., 2019), which we henceforth refer to as `LinAvg` and `RandPatch`, respectively.

Realizing training speedups through input interpolation raises interesting questions, such as how to train networks on interpolated samples, which samples to interpolate, etc. We observe that indiscriminate application of interpolation leads to a considerable drop in learning performance and model accuracy. Comparing the classification performance of networks trained with and without interpolation, we find that the efficacy of learning on interpolated inputs is adversely impacted due to interference between the processing of the constituent inputs within each interpolated input. To preserve accuracy, we therefore propose techniques to mitigate this interference.

Due to the spatial locality preserving nature of the `RandPatch` interpolation operator on account of how convolutions operate, we find that the network's internal features preserve separation between the constituent inputs within an interpolated input. However, this separation is no longer maintained in the final network layers. Based on these insights, we propose *split propagation* of interpolated inputs. The output of the final convolutional layer is separated into parts that pertain to each constituent input, and these parts are processed separately (during forward and back propagation) by the subsequent pooling and classifier layers of the network. We find that split propagation largely remedies the interference between the constituent inputs within an interpolated input. In contrast, with the `LinAvg` operator, spatial separation between the constituent inputs is not maintained. Here, we mitigate the impact of interference through *adaptive interpolation*, where we vary the weights of the constituent inputs based on their loss in previous epochs. Taken together, split propagation and adaptive interpolation vastly improve the ability of the network to learn on interpolated inputs.

Additionally, we explore applying interpolation selectively, i.e., only for a subset of training inputs in each epoch. The amenability of a training input to interpolation is measured using its classification loss. 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 interpolation. Hence, these inputs are interpolated those with moderate loss are processed without interpolation. Low-loss inputs are interpolated because their functional performance remains largely unaffected by interpolation. In contrast, we interpolate samples with high loss because a considerable percentage of such samples are unlikely to exhibit improved performance even without interpolation. For example, the ResNet50 network incurs nearly 24% Top-1 error after standard SGD training on the ImageNet dataset. A good fraction of training inputs, which occupy the higher end of the loss distribution, remain incorrectly classified through the end of training. Interestingly, interpolating such inputs allows the network to focus more on the incorrect samples with moderate loss, that actually contribute to final accuracy. We hence design a loss-driven metric to identify the training samples that are amenable to interpolation in each epoch.

Contributions: The key contributions of this work can be summarized as follows. To the best of our knowledge, InterTrain is the first effort to reduce the complexity of DNN training by combining inputs via interpolation. We propose two strategies to achieve runtime benefits from interpolation while maintaining accuracy. First, we propose split propagation and adaptive interpolation to reduce the impact of interference between the constituent inputs in an interpolated sample. Second, samples are interpolated selectively in a loss-driven manner, i.e., a loss-based amenability metric determines which samples are interpolated in epoch of epoch. It is worth noting that we add absolutely no hyper-parameters on top of baseline SGD, and that the proposed method can be combined with other training speedup techniques. For image recognition CNNs (including ResNet18/34/50 and MobileNet) trained on the ImageNet and Cifar10 datasets, we demonstrate up to 1.6× and 1.8× improvement in training time, respectively, for ∼0.2% Top-1 accuracy loss on a Nvidia RTX 2080Ti GPU.

2 Input Interpolation: Preliminaries

Input interpolation is an operation that takes multiple inputs and combines them into a composite, or interpolated, input, which combines information from each of the constituent inputs.

InterTrain utilizes two interpolation operators — `LinAvg` (Zhang et al., 2017) and `RandPatch` (Yun et al., 2019). Let us consider inputs x_1, .. x_N that are vectors comprised of scalar elements. In the case where the inputs are images, the elements represent their pixels. Both operators combine the corresponding scalar element (pixel) from each of the constituent inputs to produce an element of the interpolated input. The value of element j in the interpolated input can thus be expressed as follows:

$$X_j = \alpha_1 x_{1,j} + \ldots + \alpha_N x_{N,j} \tag{1}$$

where $\alpha_1, \ldots \alpha_N$ are in coefficients that add up to 1. The two interpolation operators differ in the further constraints placed on the values of $\alpha_1, \ldots \alpha_N$. For `LinAvg`$\alpha_1, \ldots \alpha_N$ are real-valued constants belonging to the range [0, 1]. The interpolated input is thus a linear combination of the constituent inputs with the relative values of $\alpha_1, \ldots \alpha_N$ determining the contribution from each constituent input. In contrast, `RandPatch` allows only one among $\alpha_1, \ldots \alpha_N$ to have a value of 1 for a pixel, while all others are set to 0 (i.e., the values are one-hot coded). Each pixel of the interpolated input is thus exactly equal to the corresponding pixel from one of the constituent inputs. Generally, contiguous elements in the interpolated input are selected from one randomly chosen constituent sample. This is equivalent to selecting random patches from the constituent input to form the interpolated sample. The interpolated input contains information from the constituent inputs, in a ratio determined by the relative area of the random patches. In this work, we restrict interpolation to two constituent input samples ($N = 2$) and combine them in a ratio of r : 1 - r with $r \in$ [0,1]. Thus, for `LinAvg`, α_1 = r and α_2 = 1 - r while for `RandPatch`, the ratio of number of pixels with $\alpha_1 = 1$ and $\alpha_2 = 1$ is $r/(1-r)$.

Consider constituent samples (x_1, y_1) and (x_2, y_2) interpolated to produce interpolated input X. Here, y_1 and y_2 are the target labels of x_1 and x_2 respectively. The loss of the constituent input is expressed using Equation 2, where $Loss$ refers to categorical cross-entropy loss.

$$Loss(x_i, y_i) = Loss(X, y_i) \tag{2}$$

During training, the loss of the interpolated input, which takes into account the relative contributions of each constituent input as given by the following equation, is minimized:

$$Loss(X) = r * Loss(X, y_1) + (1 - r) * Loss(X, y_2) \tag{3}$$

Input interpolation has previously been applied for data augmentation, wherein randomly selected training input samples are combined through different interpolation operators (Zhang et al., 2017; Yun et al., 2019; Verma et al., 2019; Kim et al., 2020) and added to the training set. Training on the randomly combined input samples has a regularization effect, 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 interpolation remains the same. Further, in order to realize improvements in accuracy, these techniques often require 2-3$\times$ more training epochs than baseline SGD (Yun et al., 2019; Zhang et al., 2017; Kim et al., 2020).

3 InterTrain: Accelerating DNN Training via Input Interpolation

The key idea in InterTrain is to improve the overall training time by dynamically applying the interpolation operators, `LinAvg` and `RandPatch`, on the training dataset D, to reduce the number of samples in each epoch. However, naive interpolation, e.g., where random pairs of input samples are interpolated in each training epoch to reduce the number of training samples by half, negatively impacts classification accuracy (Fig. 1(a)). The following subsections discuss the two key strategies that are critical to the overall success of InterTrain, namely, reducing the impact of interference between constituent inputs and selective interpolation.

3.1 Reducing Impact of Interference when Training with Interpolated Inputss

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

We begin by analyzing the ability of a network trained with interpolation to correctly classify the constituent inputs of an interpolated sample. At different stages of training (different training epochs), we identify the set of training samples that the network classifies correctly without

interpolation. These inputs are then interpolated in pairs ($r = 0.5$), and the network accuracy with the use of interpolation on these inputs is recorded. Five such runs are conducted to allow for different random input combinations and the results are averaged and presented in Fig. 1. Surprisingly, when interpolation is applied, the network correctly classifies less than half of the inputs that were correctly classified without interpolation (green and blue dotted curves in Fig. 1(b)), even in the final epochs of training. On further investigation, it is found that for many interpolated inputs, the network is able to correctly classify only one of the constituent inputs. The golden label of the other constituent input often does not appear even amongst the Top-5 predictions made by the network (further quantitative details in Sec. 7.4). This leads to increased loss for one of the constituent samples, consequently impacting training performance and the final validation accuracy[1]. It is thus critical to develop techniques that effectively learn on all constituent samples of an interpolated input. We next describe our approach to addressing this challenge.

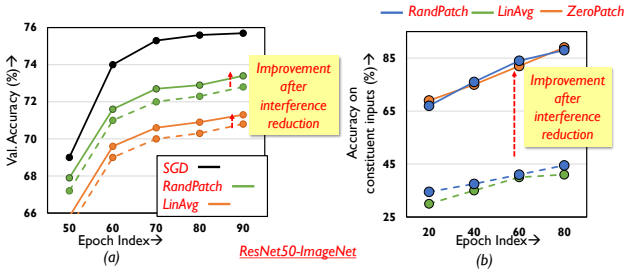

Figure 1: Classification performance with interpolation

Split Propagation: We identify two factors that contribute to the poor classification accuracy of an interpolated input's constituent inputs in the case of `RandPatch` interpolation. 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 interpolated 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 interpolation, random patches amounting to half the input area are zeroed-out. As shown using the solid orange curve (ZeroPatch) in Fig. 1(b), the drop in accuracy is only $\sim$16%, and is significantly lower compared to actual interpolation. This indicates that it is the interference between the constituent inputs that is the primary factor causing degradation in classification performance.

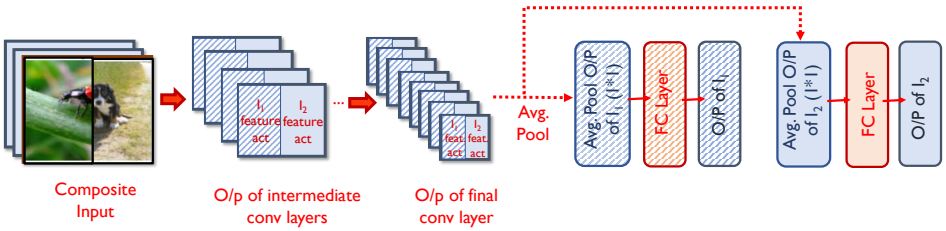

Figure 2: Training Interpolated Inputs

Examining the intermediate representations of the network while processing interpolated inputs sheds some light on this interference. By virtue of the nature of convolutions, the spatial separation between constituent inputs in the interpolated input is maintained through many layers of the network. For example, in Fig. 2, the right half of the features in the final convolution layer's output pertain to the right half of the interpolated 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, we observe that the fully connected layer correctly classifies only one of the constituent inputs.

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 Fig. 2, we identify the region in the final convolutional layer's output maps pertaining to each constituent input, and pass the features separately through the remaining layers of the network. Both constituent inputs of interpolated sam-

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

ples are now classified correctly, leading to a significant improvement in classification performance (solid blue curve in Fig. 1(b)). 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 (Fig. 1(a)). 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.

Adaptive Interpolation: Unlike `RandPatch`, the `LinAvg` operator averages each element of the constituent inputs prior to processing 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 Fig. 1(a) that the validation accuracy with `LinAvg` is even lower compared to `RandPatch` , due to a slower rate at which training loss improves for the interpolated inputs. Naturally, a simple boost in performance can be achieved by atleast improving the loss for one of the constituent inputs of the interpolated input. We thus adapt the weight (r) of constituent inputs so as to favour the more difficult input, as identified by the loss in the previous epoch. Specifically, as shown below, we set r as the ratio of the losses of the constituent samples in the previous epoch. As seen in Fig. 1(a), this provides a boost in classification accuracy.

$$r_{E+1} = \frac{\text{Loss of constituent input 1 in epoch E}}{\text{Loss of constituent input 2 in epoch E}}$$

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

3.2 Selective Interpolation

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

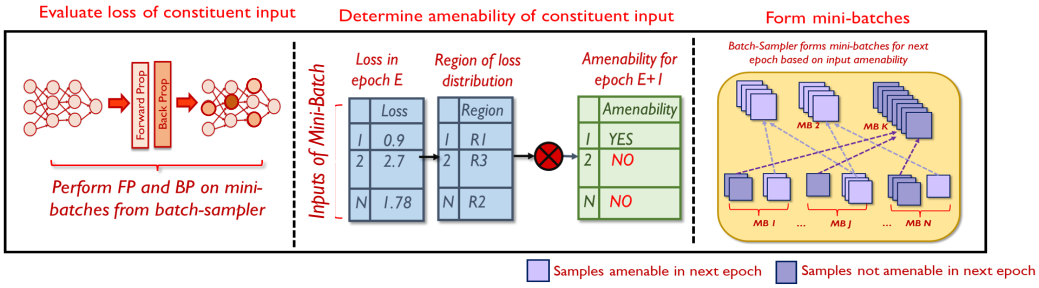

Figure 3: Overview of Selective Interpolation

Overview: The proposed selective interpolation strategy consists of three steps as shown in Fig. 3. 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 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 interpolation in the next epoch E+1, subsequent to which it is added appropriately to S_{int} or S_{noInt}. Finally, the batch-sampler forms mini-batches for the epoch E+1 by randomly drawing samples from either S_{int} or S_{noInt}.

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

3.2.1 EVALUATING AMENABILITY TO INTERPOLATION IN EPOCH E

A suitable loss-based amenability metric must estimate the subsets S_{int} and S_{noInt} 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 interpolation, at different stages of the training process.

Consider models trained with `LinAvg` and `RandPatch` at three different training epochs as shown. At each selected epoch, we compute the L_1 difference of the loss of every sample A with and without interpolation, i.e., $loss_{int}(A)$ and *loss(A)* respectively. $loss_{int}(A)$ is computed using Equation 2. We observe that $loss_{int}(A)$ deviates and increases further away as *loss(A)* increases, consistently across the benchmarks analyzed for both operators (Fig. 4(a) depicts the same for `RandPatch`). In other words, the graph indicates that *as loss(A) increases, its amenability to interpolation decreases*. Furthermore, we inspect the samples in the low loss regime prior to interpolation, and find that a majority are correctly classified. From Fig. 4(b) for the `RandPatch` operator it is seen that their accuracy is reasonably maintained after interpolation as well.

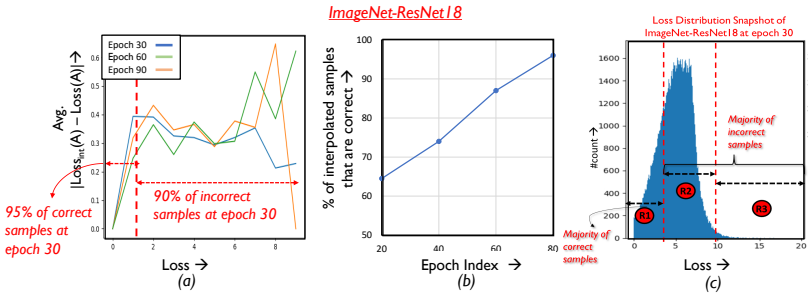

Figure 4: Analyzing amenability to interpolation

Hence, for samples that are not interpolated in epoch E, we determine their amenability to interpolation in the next epoch based on the particular region of the loss distribution it belongs to[2]. As illustrated in Fig. 4(c), the loss distribution is divided into three regions that utilize a different criteria for gauging amenability. We now discuss the criteria for each region, and the conditions for continuing interpolation in subsequent epochs.

Region 1 corresponds to the area in the loss distribution where a majority of the correctly classified samples are located. From Fig. 4(b) we know that the loss, and to a certain extent the classification accuracy of such samples remains largely unaffected by interpolation and are hence interpolated 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 interpolating 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 interpolating 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_{noInt} respectively (calculated from epoch 0 to E-1), and let L_{mid} denote the average of the two quantities, i.e.,

$$L_{mid} = 0.5 * (L_{corr} + L_{incorr}) \tag{4}$$

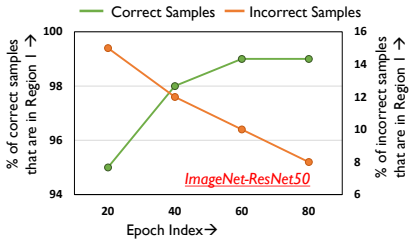

Figure 5: Efficacy of threshold L_{mid}

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. Fig. 5 plots the efficacy of L_{mid} across different epochs

[2]In Sec. 7.5, we find that classification loss does not change rapidly across epochs. Sample amenability in epoch E+1, can thus be determined based on classification loss in epoch E

(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 ($< 10\%$). In fact, we find that interpolating correct samples with loss $> L_{mid}$ has an adverse effect on accuracy, as they represent outliers in the distribution. 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 using other techniques, such as 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.

Amenability Criteria for Region 1: Consider a sample A belonging to Region 1 in epoch E, i.e., $Loss_A < L_{mid}$. From Figure 4(b) it is known that samples in Region 1 are likely to be correctly classified prior to interpolation. We interpolate 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_{noInt}. Fig. 6 illustrates the temporal variation in the number of samples that are in S_{int}, 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. The graph also depicts the fraction of samples that deflect to S_{noInt} every epoch, which is a very small fraction of the samples that are interpolated. This justifies the design of the amenability rule for Region 1.

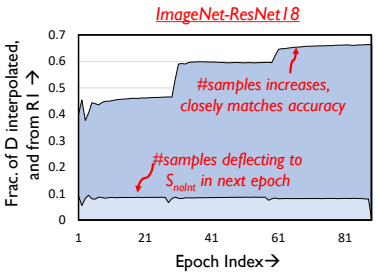

Figure 6: Amenability of Region 1

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 they are trained without interpolation. 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 Fig. 7(a). 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 interpolation.

Some challenges however persist. Essentially, we cannot naively interpolate all samples in Region 3. Furthermore, classification statistics evolve during training rendering it difficult to determine which samples to interpolate 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 Fig. 7(b), 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 thus stipulate criteria to identify the desired subset of Region 3 samples, and interpolate these only for some epochs. In addition to belonging to Region 3, if a sample's loss increases over consecutive epochs (i.e., become increasingly difficult) it is interpolated for the next epoch, ensuing which it is brought back to S_{noInt}. In Fig. 8(b), we find that increasing the period of time k for which the difficult samples must exhibit increasing loss and subsequently be interpolated, 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.

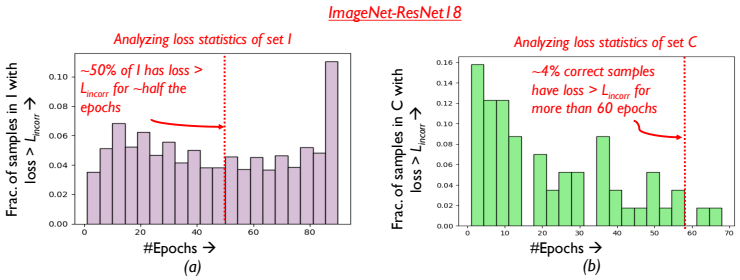

Figure 7: Loss dynamics of samples in set I and set C

The temporal variation in the fraction of Region 3 samples interpolated every epoch is depicted in Fig. 8(a). This fraction decreases across epochs, since several samples in Region 3 shift to Region 1 as accuracy improves. Interestingly, interpolating 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 interpolation. 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 interpolating such difficult samples instead of skipping them in Sec. 7.8.

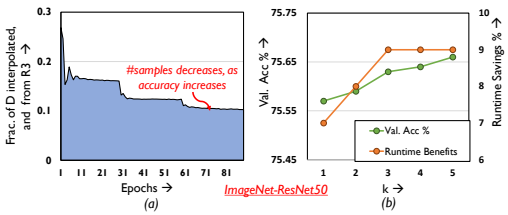

Figure 8: Amenability for samples in Region 3

Determining sample amenability every epoch adds not more than 2% overhead in runtime on average. The proposed amenability criteria thus help us successfully realize selective interpolation, i.e., achieve a competitive runtime efficiency versus accuracy trade-off. In Sec. 7.7 we underscore the efficacy of our criteria by comparing against different metrics.

4 Experimental Results

In this section, we showcase the runtime benefits achieved by InterTrain across a suite of image-recognition DNNs, *viz.* ResNet18, ResNet34, ResNet50 (He et al., 2015) and MobileNetV2 (Sandler et al., 2018), for the ImageNet dataset (Deng et al., 2009). All experiments are conducted on a server with 4 Nvidia RTX 2080Ti GPUs with the batch size set to 64 per GPU, unless otherwise mentioned. We focus on the ImageNet results in this section. Our Cifar10 results, experimental methodology details, and extensive quantitative comparisons and ablation studies are presented Section 7.

4.1 Execution Time Benefits

Table 1 presents the training performance of baseline SGD and InterTrain on different benchmarks in terms of the Top-1 classification error and speed-up. On average, across all benchmarks, InterTrain interpolates nearly 48% and 68% of the training dataset per epoch with `LinAvg` and `RandPatch` respectively. As can be seen, `RandPatch` achieves a slightly superior trade-off than `LinAvg` across all benchmarks, achieving upto $1.6\times$ reduction in runtime compared to to the baseline, while sacrificing only $\sim$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 interpolated. Fig. 10 in Sec. 7.2 illustrates the validation curves for the ResNet benchmarks when trained with InterTrain using the `RandPatch` operator and baseline SGD. InterTrain clearly outperforms the baseline, consistently achieving better accuracy at iso-runtime across training.

Table 1: ImageNet

Network	Training Strategy	Top-1 Error	Speed-Up
ResNet18	Baseline SGD	30.2%	1×
	InterTrain-**`RandPatch`**	**30.44%**	**1.51**×
	InterTrain-**`LinAvg`**	**30.6%**	**1.32**×
ResNet34	Baseline SGD	26%	1×
	InterTrain-**`RandPatch`**	**26.25%**	**1.54**×
	InterTrain-**`LinAvg`**	**26.4%**	**1.37**×
ResNet50	Baseline SGD	24.3%	1×
	InterTrain-**`RandPatch`**	**24.45%**	**1.56**×
	InterTrain-**`LinAvg`**	**24.6%**	**1.41**×
MobileNetV2	Baseline SGD	28.5%	1×
	InterTrain-**`RandPatch`**	**28.76%**	**1.52**×
	InterTrain-**`LinAvg`**	**29%**	**1.3**×

4.2 Quantitative Comparison Study

We compare the performance of InterTrain against competing state-of-the-art methods, depicted in Fig. 9. We consider representative works from different categories of techniques that broadly explore input approximations for training acceleration, specifically (i) instance skipping strategies, and (ii) training with reduced resolution. Additional comparisons against some model size reduction efforts are provided in Sec. 7.10.

Instance Skipping: In these techniques samples that the network finds easy to classify, as identified by low classification loss, are skipped during later epochs, thereby resulting in fewer mini-batches as training proceeds. As a representative of instance-skipping, we specifically consider AutoAssist (Zhang et al., 2019). Two issues are typically encountered by instance skipping techniques. First, as no training is conducted on the samples that are skipped every epoch, 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. InterTrain achieves better model accuracy and runtime benefits as the network is ultimately trained on every input in each epoch. This allows us to effectively reduce the minibatches more aggressively compared to instance skipping, while incurring negligible overheads incurred to form S_{int} and S_{noInt}. Further, interpolation allows us to extract additional benefits by approximating training on Region 3 samples, which instance skipping strategies struggle to leverage (Sec. 7.8). In Sec. 7.9 we show that even when certain overheads associated with instance skipping strategies are overlooked, InterTrain still achieves a superior trade-off.

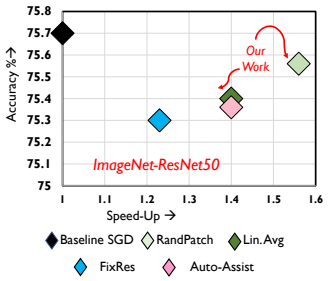

Figure 9: Comparison against existing efforts

Reducing Input Resolution: Input resolution has a direct impact on training runtime, with the latter increasing with input resolution. However, in FixRes (Touvron et al., 2020), it is found that smaller inputs result in poorer activation statistics. To restore accuracy, several epochs of fine-tuning are performed after baseline SGD, thereby affecting runtime savings. At similar runtime, we clearly achieve better accuracy, as illustrated in Figure 9.

5 Related Work

This section discusses related research efforts to accelerate DNN training.

Hyper-parameter tuning: Many notable algorithmic efforts are directed towards achieving training efficiency by controlling the hyper-parameters involved in gradient-descent, notably the learning rate and momentum. (You et al., 2017a; Akiba et al., 2017; Goyal et al., 2017; You et al., 2017b) propose learning rate tuning algorithms that achieve training in less than an hour with no loss in accuracy, when distributed to over hundreds of CPU/GPU cores.

Input and model size reduction during training: Training runtimes are proportional to the input resolution, due to larger activations observed across the network. In (Touvron et al., 2020), the authors propose fine-tuning strategies that enable training at reduced resolution, while minimizing the loss in accuracy caused due to smaller activation sizes. In contrast, model size reduction investigates dynamically pruning (Yuan et al., 2020) or quantizing (Sun et al., 2019) a model during training itself. Training a reduced-capacity model, or with lower-precision results in training speed-ups. Taking a slightly different approach (Huang et al., 2016) proposes stochastically dropping residual blocks on extremely deep networks such as ResNet-1202, not only for training runtime benefits but also better accuracies due to improved gradient strength.

Instance skipping strategies : Recent research efforts have discovered that not all training samples are required for improving loss minimization during SGD training (Jiang et al., 2019; Zhang et al., 2019). That is, a sizable fraction of the samples can be skipped during several epochs, depending on their impact on the classification loss evaluated during . This translates to a reduction in mini-batches, providing considerable runtime benefits.

6 Conclusion

In this paper, we introduce a new approach to improve the training efficiency of state-of-the-art DNNs by utilizing input interpolation. We propose InterTrain that comprises of two strategies to achieve an acceptable accuracy versus speed-up trade-off. First, we propose split propagation and adaptive interpolation to reduce the impact of interference between the constituent inputs in an interpolated sample. Second, we apply interpolation selectively, i.e., only on a subset of the training dataset every epoch. Across DNNs trained on the Cifar10 and ImageNet datasets, we achieve upto a 1.8× and 1.6× improvement in runtime respectively for around 0.2% loss in Top-1 accuracy.

REFERENCES

Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch SGD: training resnet-50 on imagenet in 15 minutes. *CoRR*, abs/1711.04325, 2017. URL http://arxiv.org/abs/1711.04325.

Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. Training costs, 2018. URL {https://openai.com/blog/ai-and-compute/}.

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

Yoav Goldberg and Graeme Hirst. *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017. ISBN 1627052984.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016. URL http://arxiv.org/abs/1603.09382.

Angela H. Jiang, Daniel L. K. Wong, Giulio Zhou, David G. Andersen, Jeffrey Dean, Gregory R. Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C. Lipton, and Padmanabhan Pillai. Accelerating deep learning by focusing on the biggest losers, 2019.

Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL https://doi.org/10.1145/3065386.

Duo Li, Aojun Zhou, and Anbang Yao. Hbonet: Harmonious bottleneck on two orthogonal dimensions. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Mattan Erez, and Sujay Shanghavi. Prunetrain: Gradual structured pruning from scratch for faster neural network training. *CoRR*, abs/1901.09290, 2019. URL http://arxiv.org/abs/1901.09290.

Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *CoRR*, abs/1503.08909, 2015. URL http://arxiv.org/abs/1503.08909.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. In *NeurIPS*, 2019.

Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. *CoRR*, abs/2104.00298, 2021. URL https://arxiv.org/abs/2104.00298.

Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy, 2020.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, Aaron Courville, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states, 2019.

Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for imagenet training. *CoRR*, abs/1708.03888, 2017a. URL http://arxiv.org/abs/1708.03888.

Yang You, Zhao Zhang, Cho-Jui Hsieh, and James Demmel. 100-epoch imagenet training with alexnet in 24 minutes. *CoRR*, abs/1709.05011, 2017b. URL http://arxiv.org/abs/1709.05011.

Xin Yuan, Pedro Savarese, and Michael Maire. Growing efficient deep networks by structured continuous sparsification. *CoRR*, abs/2007.15353, 2020. URL https://arxiv.org/abs/2007.15353.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. *CoRR*, abs/1905.04899, 2019. URL http://arxiv.org/abs/1905.04899.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017. URL http://arxiv.org/abs/1710.09412.

Jian Zhang and Ioannis Mitliagkas. Yellowfin and the art of momentum tuning, 2018.

Jiong Zhang, Hsiang-Fu Yu, and Inderjit S. Dhillon. Autoassist: A framework to accelerate training of deep neural networks. *CoRR*, abs/1905.03381, 2019. URL http://arxiv.org/abs/1905.03381.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630, 2015. URL http://arxiv.org/abs/1511.08630.

7 APPENDIX

7.1 EXPERIMENTAL SETUP

This subsection describes the experimental setup used for realizing the baseline and proposed training schemes, on the benchmarks specified in Section 4 of the main paper. We conduct our experiments on the complete training and test datasets of each benchmark, using the PyTorch (Paszke et al., 2019) framework.

Baseline: We consider SGD training as the baseline in our experiments. The hyper-parameters used in SGD training of each of the benchmarks are described below.

ImageNet: For experiments in Section 4.1 we utilize a batch-size of 64 per GPU, for all benchmarks. For the ResNet18, ResNet34 and ResNet50 benchmarks the initial learning rate set to 0.025. The learning rate is decreased by 0.1 every 30 epochs, for a total training duration of 90 epochs, and the weight decay is $4e-5$. The MobileNetV2 benchmark utilizes an initial learning rate of 0.0125. We use a cosine learning rate decay schedule, as in (Li et al., 2019) for 150 epochs. The weight decay is set to $4e-5$. All benchmarks use an input size of 224*224*3.

Cifar10: All Cifar10 experiments utilize a batch-size of 64. The Cifar10 benchmarks are trained with an initial learning rate of 0.05 that is decayed by 0.1 every 10 epochs, across 90 epochs. The weight decay is set to $5e-4$. All benchmarks utilize an input size of 32*32*3.

InterTrain: InterTrain uses the same learning rate, weight decay, and number of epochs as baseline SGD, requiring no additional hyper-parameters. We use the same random seed for both our baseline and InterTrain experiments. Results in Sec. 4.1 are reported by averaging over 3 different training runs.

7.2 Validation Accuracy curves for ImageNet

In this sub-section we illustrate the validation accuracy curves for InterTrain and compare the same against baseline performance. Here, we have normalized the runtime to that of InterTrain on each benchmark. Clearly, InterTrain achieves better accuracy for the same runtime across all epochs of training. For example, in Fig. 10(b), we highlight that InterTrain completes epoch 30 far earlier than baseline SGD.

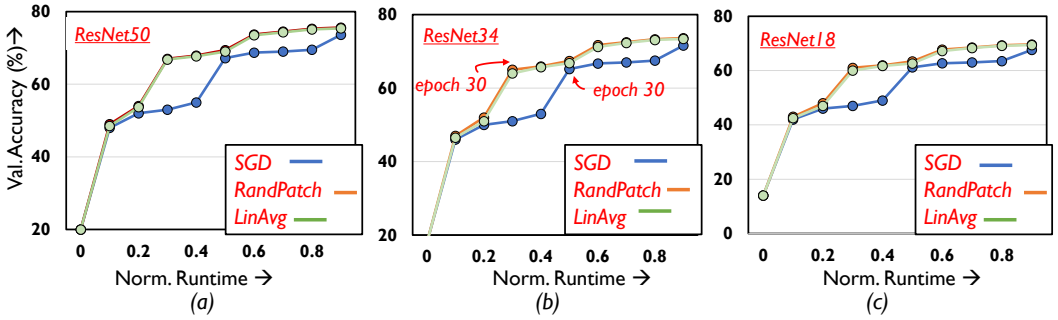

Figure 10: Validation Accuracy curves

7.3 Experimental Results on Cifar10

Table 2: Cifar10

Network	Training Strategy	Top-1 Error	Speed-Up
ResNet18	Baseline SGD	6.5%	1×
	InterTrain-**RandPatch**	**5.4%**	**1.74**×
	InterTrain-**LinAvg**	**5.7%**	**1.69**×
ResNet34	Baseline SGD	5.2%	1×
	InterTrain-**RandPatch**	**4.2%**	**1.78**×
	InterTrain-**LinAvg**	**4.6%**	**1.71**×

To underscore the wide applicability of InterTrain, we present our runtime and accuracy trade-off achieved on the Cifar10 benchmarks in Table 2. Across our benchmarks, `LinAvg` achieves upto 1.7 × improvement in runtime, while `RandPatch` achieves a 1.8× runtime improvement. Both techniques provide upto a 0.8-1% boost in accuracy, due to the imrpoved regularization provided via interpolating samples.

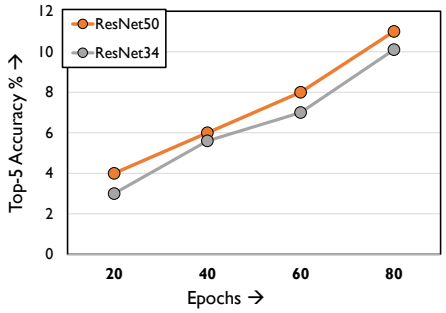

Figure 11: Top-5 classification accuracy of second constituent input

7.4 Analysis of Top-5 accuracy without interference reduction

In Sec. 3.1 it is mentioned that the network is unable to detect both constituent inputs when interference between them is not reduced. At most, the network detects only one of the constituent inputs, with the second constituent rarely appearing in the Top-5 predictions made. We provide the Top-5 classification accuracy of the second constituent in Fig. 11 for the `RandPatch` operator, prior to reducing interference - clearly, accuracy never exceeds 12%. This emphasizes the need for devising strategies to reduce interference between the constituent inputs of a composite sample.

7.5 Analysis of loss across consecutive epochs

As mentioned in Sec. 3.2, we utilize the loss of a sample in epoch E to determine it's amenability to interpolation in epoch E+1. However, several mini-batches pass before a sample is trained again in the next epoch. As the model undergoes many changes to its weights, it is possible that the loss of a sample in epoch E might be quite substantially different from that in epoch E+1.

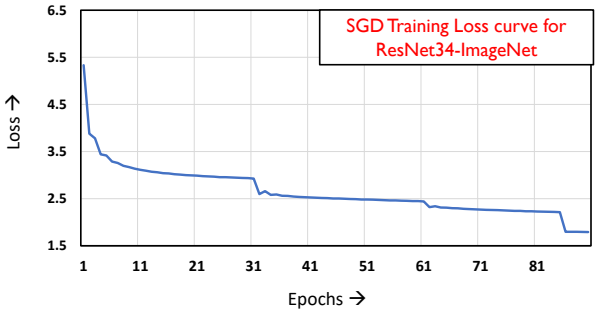

Figure 12: Change in average loss across epochs

Fig. 12 plots the loss curve averaged across all training examples when trained with baseline SGD. The loss appears to change rapidly only for the first few epochs, and later when the learning rate changes. In other periods, changes in loss happen more gradually. We find that the same analysis is generally applicable when samples are interpolated as well. This thus justifies using the loss in epoch E to justify amenability in epoch E+1.

7.6 Ablation Study

In this subsection we conduct an ablation analysis of InterTrain.

Contribution of key strategies: InterTrain uses two strategies to achieve an optimal accuracy versus runtime trade-off, i.e., reducing impact of interference between constituent inputs and selective interpolation. Fig. 13(a) depicts the contribution of each strategy towards runtime savings, for the `RandPatch` operator. The light blue markings indicate naive interpolation. Selective interpolation automatically identifies a subset of training samples that can be interpolated every epoch such that

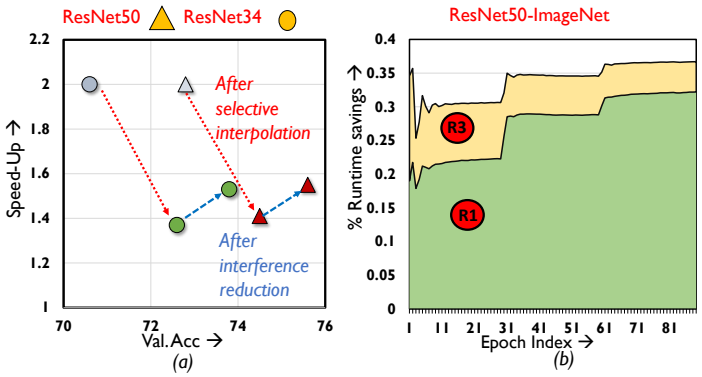

Figure 13: Ablation analysis

classification accuracy is not impacted severely. However, if interference between the constituent inputs is not mitigated, training performance on interpolated samples is poor (green markings). Consequently, the selective interpolation strategy is forced to become conservative, identifying fewer samples that can be interpolated every epoch without affecting accuracy severely. Reducing interference between the constituent inputs improves both accuracy by more than 1%, and speed-up by 10% (red markings).

Breakdown of selective interpolation: We breakdown selective interpolation by examining the region of the loss distribution that provides the most benefits. From Fig. 13(b) (generated using `RandPatch`) 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. Interpolating Region 3 samples, accounts for additional 8% runtime savings.

7.7 Analyzing efficacy of amenability metric

As part of our key strategies to achieve a good accuracy versus runtime efficiency trade-off, we propose selectively interpolating samples based on their amenability to interpolation in Section 3.2. We take into consideration the region of the loss distribution where the sample occurs in epoch E to appropriately decide whether the sample should be interpolated in the next epoch. Each region differs based on the estimated impact interpolating a sample may have on accuracy. Consequently, each region has its own criteria for gauging amenability for the next epoch.

In Table 3, we compare our proposed selective interpolation strategy against the following set of rules to gauge amenability, using the `RandPatch` operator.

- First, we analyze the trade-off achieved when we interpolate inputs that were correctly classified in previous epoch, instead of using L_{mid} as in Sec. 3.2. Essentially, only those samples that are correct in epoch E are interpolated in the next epoch. We find that the L_{mid} threshold (Row 1) achieves slightly better classification accuracy, as outlier inputs with correct classification are avoided.
- Next, we compare against an average loss threshold, i.e., we calculate the running average of the loss across all the samples in S_{noInt}. If a sample in epoch E has loss lower than the running average, it is interpolated in the next epoch and vice-versa. As can be seen, our L_{mid} threshold (Row 1) achieves better speed-ups for nearly the same classification accuracy. Across epochs, classification accuracy improves and average loss reduces, often with several correctly classified samples with loss above the average loss. This metric thus loses the opportunity to approximate training effort on these these correctly classified samples that are amenable to interpolation.
- Finally, we compare the efficacy of our Region 3 criterion. We observe the trade-off achieved when all samples above L_{incorr} are interpolated, in addition to Region 1 approximations. Clearly, our proposed criterion attains better accuracy (Row 2).

Table 3: Analyzing efficacy of our amenability metric

Benchmark	Amenability Metric	Top-1 err	Speed-Up
ResNet50	Our Effort (Region 1 only)	24.56%	1.38
	Our Effort (Regions 1 and 3)	24.45%	1.56
	Threshold = Accuracy	24.80%	1.36
	Threshold = Average Loss	24.50%	1.33
	Region 1 and threshold = L_{incorr} for Region 3	25.14%	1.74

7.8 INTERPOLATING VERSUS SKIPPING DIFFICULT SAMPLES

In Sec. 3.2, we proposed the concept of interpolating difficult samples that consistently occur in Region 3 of the loss distribution. These samples are likely to stay incorrect even at the final epochs of training, and it is hence practical to reduce training effort on such samples. In this subsection, we analyze the impact on accuracy and runtime if training effort was completely eliminated on such samples, i.e., these samples are skipped for a few epochs, instead of interpolated.

We apply the same policy discussed in Sec. 3.2- if a sample in Region 3 of the distribution exhibits increasing loss across k consecutive epochs, these samples are skipped for the next k epochs after which they are trained without interpolation again. Table 4 highlights the accuracy versus runtime trade-off achieved for different values of k. For a clearer comparison, we do not apply interpolation in Region 1. Clearly, interpolating such samples is critical for achieving competitive accuracy.

Table 4: Comparing interpolation versus skipping Region 3 samples

Benchmark	k	Top-1 err (SGD)	Speed-Up (SGD)	Top-1 err (Our Effort)	Speed-Up (Our Effort)
ResNet50-ImageNet	1	24.05%	1.12	24.45%	1.08
	2	25.41%	1.14	24.4%	1.1
	3	25.74%	1.15	24.36%	1.11

7.9 COMPARATIVE ANALYSIS AGAINST INSTANCE-SKIPPING TECHNIQUES

In this sub-section, we further compare selective interpolation against selective skipping strategies.

In selective skipping strategies, a subset of the training dataset that can be skipped every epoch is identified in a manner that does not impact accuracy. This results in fewer mini-batches every epoch, thereby providing training runtime benefits. The samples discarded are typically those the network finds easy to classify, and hence need not be included throughout training. In (Jiang et al., 2019), the authors conduct forward propagation (FP) across all the training set examples every epoch. At the end of FP, the loss of all samples are computed, and those in the lower percentile of the loss distribution are skipped. Back-propagation (BP) is conducted only on the remaining samples. (Zhang et al., 2019) alleviate the time required to perform FP on the easy samples, by using a light-weight model to predict this subset every epoch- both FP and BP are now performed only on the difficult samples. However, updating the light-weight model to match the evolving characteristics of the main model trained incurs overhead.

As mentioned in Section 4, some drawbacks exist. First, the subset of samples that can be skipped is often a small fraction of the dataset, so that accuracy is not affected. Second, additional overheads are incurred in determining which samples to skip every epoch. We replicate instance selective strategies and overlook one of the overheads, i.e., the overhead required to determine the samples that can be skipped. Interestingly, we find that our proposed effort still achieves a superior runtime versus accuracy trade-off.

Similar to (Jiang et al., 2019), we utilize FP to estimate the subset of easy samples every epoch. Specifically, the loss of every sample computed after FP is compared against some threshold to determine whether it can be skipped in the next epoch. We use L_{corr} as the threshold, as L_{mid} appears to severely impact accuracy. Importantly, we do not factor in the runtime to conduct FP on samples that are discarded in the next epoch, akin to having a zero-cost oracle predict this subset.

As shown in Tab. 5 for the ResNet50 network trained on the ImageNet dataset, our effort clearly still achieves better accuracy and runtime savings.

Table 5: Comparing oracle-based skipping against our effort

Benchmark	Training Strategy	Top-1 err (SGD)	Speed-Up (SGD)
ResNet50-ImageNet	Selective Interpolation	24.45%	1.56
	Instance Skipping	25.12%	1.44

This can be chalked up to the fact that in InterTrain, some form of training is conducted across all the samples in the dataset, resulting in better accuracy. In turn, this allows us to approximate training effort on a larger fraction of samples (the green columns in Fig. 14 always remain shorter than the yellow columns across all epochs). Furthermore, as discussed in the previous sub-section, instance-skipping techniques are unable to leverage the classification performance and runtime benefits accrued by skipping extremely difficult samples , i.e., Region 3 approximations. This experiment underscores the importance of our effort towards accelerating DNN training. Interpolating a subset of the training dataset every epoch thus provides a cheaper training alternative, without sacrificing accuracy.

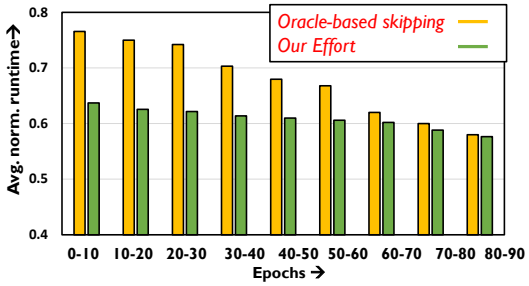

Figure 14: Comparing oracle-based skipping against our effort

7.10 Comparison against additional existing efforts

In this subsection we compare InterTrain against some additional training acceleration techniques, i.e., model size reduction techniques. We illustrate the same in Fig. 15.

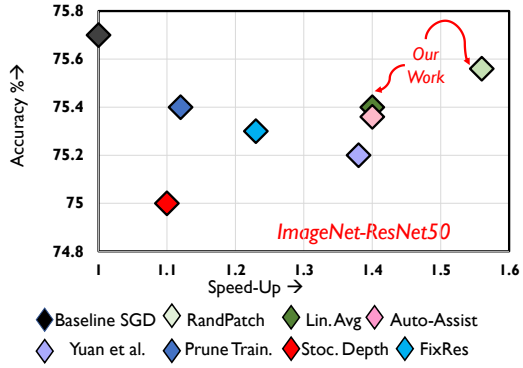

Figure 15: Comparison against existing efforts

We first analyze StocDepth (Huang et al., 2016), wherein the authors stochastically bypass residual blocks by propagating input activations/error gradients via identity or downsampling transformations, thereby providing better runtime. However, the approach is targeted towards extremely deep

networks, and incurs a noticeable accuracy loss on smaller networks such as ResNet50. (Yuan et al., 2020) explores pruning during training by reducing the size of the weight and activation tensors in a structured manner, providing speed-ups on GPU/TPU platforms. On complex benchmarks such as ResNet50, such techniques incur a significant drop in accuracy.