

# ACCELERATING DEEP LEARNING BY FOCUSING ON THE BIGGEST LOSERS

Angela H. Jiang<sup>1</sup> Daniel L.-K. Wong<sup>1</sup> Giulio Zhou<sup>1</sup> David G. Andersen<sup>1,2</sup> Jeffrey Dean<sup>2</sup>  
 Gregory R. Ganger<sup>1</sup> Gauri Joshi<sup>1</sup> Michael Kaminsky<sup>3,1</sup> Michael Kozuch<sup>4</sup> Zachary C. Lipton<sup>1</sup>  
 Padmanabhan Pillai<sup>4</sup>

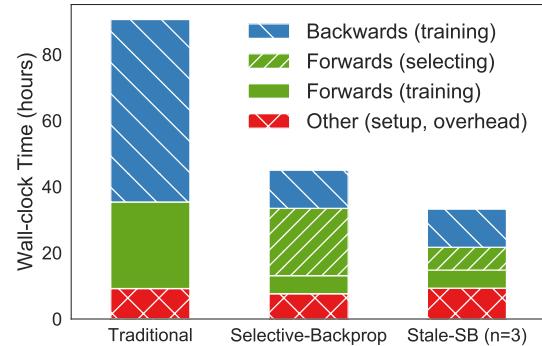
## ABSTRACT

This paper introduces Selective-Backprop, a technique that accelerates the training of deep neural networks (DNNs) by prioritizing examples with high loss at each iteration. Selective-Backprop uses the output of a training example’s forward pass to decide whether to use that example to compute gradients and update parameters, or to skip immediately to the next example. By reducing the number of computationally-expensive back-propagation steps performed, Selective-Backprop accelerates training. Evaluation on CIFAR10, CIFAR100, and SVHN, across a variety of modern image models, shows that Selective-Backprop converges to target error rates up to 3.5x faster than with standard SGD and between 1.02–1.8x faster than a state-of-the-art importance sampling approach. Further acceleration of 26% can be achieved by using stale forward pass results for selection, thus also skipping forward passes of low priority examples. The implementation of Selective-Backprop is open-source and can be found at <https://bit.ly>SelectiveBackpropAnon>.

## 1 INTRODUCTION

While training neural networks (e.g., for classification), computational effort is typically apportioned equally among training examples, regardless of whether the examples are already scored with low loss or if they are mis-predicted by the current state of the network (Hinton, 2007). In practice, however, not all examples are equally useful. As training progresses, the network begins to classify some examples accurately, especially redundant examples that are well-represented in the dataset. Training using such samples may provide little to no benefit; hence, limited computational

<sup>1</sup>Carnegie Mellon University <sup>2</sup>Google Brain <sup>3</sup>BrdgAI  
<sup>4</sup>Intel Labs. Correspondence to: Angela H. Jiang <[aahjiang@cs.cmu.edu](mailto:aahjiang@cs.cmu.edu)>.



**Figure 1:** Comparison and breakdown of training time by Traditional training and proposed Selective-Backprop approaches, for training Wide-Resnet on SHVN until 1.72% error rate is achieved (1.2 times the final error of Traditional). SB accelerates training by reducing the number of computationally expensive backward passes. Stale-SB further accelerates training by sometimes reusing losses calculated in the prior epoch for example selection.

resources may be better spent training on examples that the network has not yet learned to predict correctly.

Figure 2 illustrates the redundancy difference between “easy” examples and “hard” examples. Figure 2a shows examples from CIFAR10 that consistently produce low losses over the course of training. Compared with examples that generate high losses (Figure 2b), the classes of low-loss examples are easily distinguishable.

Motivated by the hinge loss (Rosasco et al., 2004), which provides zero loss whenever an example is correctly predicted by sufficient margin, this paper introduces Selective-Backprop (SB), a simple and effective sampling technique for prioritizing high-loss training examples throughout training. We suspect, and confirm experimentally, that examples with low loss correspond to gradients with small norm and thus contribute little to the gradient update. Thus, Selective-Backprop uses the loss calculated during the forward pass as a computationally cheap proxy for the gradient norm, enabling us to decide whether to apply an update without having to actually compute the gradient. Selective-Backprop

prioritizes gradient updates for examples for which a forward pass reveals high loss, probabilistically skipping the backward pass for examples exhibiting low loss.

By reducing computation spent on low-loss examples, Selective-Backprop reaches a given target accuracy significantly faster. Figure 1 shows this effect for one experiment in our evaluation. As seen in the first stacked bar (“Traditional”), in which every example is fully trained on in each epoch, backpropagation generally consumes approximately twice the time of forward propagation (Chintala). In this experiment, Selective-Backprop (second stacked bar) reduces the number of backpropagations by  $\approx 70\%$  and thereby cuts the overall training time in half. Across a range of models and datasets, our measurements show that Selective-Backprop speeds training to target errors by up to 3.5x.

Given Selective-Backprop’s reduction of backpropagations, over half of the remaining training time is spent on forward passes, most of which correspond to non-selected examples. These forward passes do play an important role, though, because the loss order of examples varies throughout training and varies more with Selective-Backprop. A model might generate relatively low loss on a given example after some training, but progressively higher losses on the same example if it is ignored for several epochs (Hinton, 2007). Selective-Backprop evaluates sampling probabilities on the basis of an up-to-date forward pass, ensuring its assessment of the network’s performance on the example is out of date.

The number of forward passes can be reduced, however, by allowing for some staleness during selection. One simple approach, for which we call the corresponding SB variant Stale-SB (third stacked bar), is to perform forward passes to inform selection only every  $n^{\text{th}}$  epoch. In intervening epochs, Stale-SB uses results of the most recent forward pass of an example for selection, though it needs an up-to-date forward pass for the training of selected examples. For  $n=3$ , Figure 1 shows that Stale-SB avoids approximately half of all forward passes, reducing training time by 26% relative to Selective-Backprop with minimal loss in final accuracy. Although our experiments show Stale-SB captures most of the potential reduction, we also discuss other approaches to reducing selection-associated forward pass time.

Selective-Backprop requires minimal modifications to existing training protocols, applies broadly to DNN training, and works in tandem with data augmentation, cutout, dropout, and batch normalization. Our experiments show that, without changing initial hyperparameters, Selective-Backprop and Stale-SB can decrease training times needed to achieve target error rates. Across a wide range of configuration options, including training time budgets, Selective-Backprop and Stale-SB provide most of the Pareto-optimal choices. Sensitivity analyses also show that Selective-Backprop is robust to label error and effective across a range of selectivi-

ties.

This paper makes three primary contributions: (1) The design and evaluation of Selective-Backprop and Stale-SB, practical and effective sampling techniques for deep learning; (2) Measurements showing that, compared to traditional training, Selective-Backprop and Stale-SB reduce training times to target errors on CIFAR10, CIFAR100, and SVHN by up to 3.5x and 5x, respectively; and (3) Comparison to a state-of-the-art importance sampling approach introduced in (Katharopoulos & Fleuret, 2018), showing that Selective-Backprop and Stale-SB reduce training times needed to achieve target accuracy by 1.02–1.8x and 1.3–2.3x.

## 2 RELATED WORK

Several papers propose to reduce variance and accelerate neural network training. The key idea of these techniques is to bias the selection of examples from the training set, selecting some examples with higher probability than others. A common approach is to use importance sampling, where the goal is to more frequently sample rare examples that might correspond to large updates. Classic importance sampling techniques weight the items inversely proportional to the probability that they are selected, producing an unbiased estimator of the stochastic gradient. Previous approaches use importance sampling to reduce the number of training steps to reach a target error rate in classification (Gao et al., 2015; Johnson & Guestrin, 2018; Loshchilov & Hutter, 2015), reinforcement learning (Schaul et al., 2016), and object detection (Lin et al., 2017; Shrivastava et al., 2016). These works generate a distribution over a set of training examples and then train a model by sampling from that distribution with replacement. They maintain historical losses for each example, requiring at least one full pass on all data to construct a distribution, which they subsequently update over the course of multiple training epochs. Consequently, these approaches must maintain additional state proportional to the training set in size and rely on hyperparameters to modulate the effects of stale history. In contrast, the base Selective-Backprop approach does not require such state, though some optimization options do.

The approach most related to ours (Katharopoulos & Fleuret, 2018) also removes the requirement to maintain history, providing a fully-online approach to importance sampling for classification. Similar to Selective-Backprop, it uses extra forward passes instead of relying on historical data, allowing it to scale more easily to large datasets, and it also makes decisions based on an up-to-date state of the network. Their sampling approach, however, predetermines the number of examples selected per batch, so example selection is dictated by the distribution of losses in a batch. It also relies on a variable starting condition. We compare against this technique in Section 5.

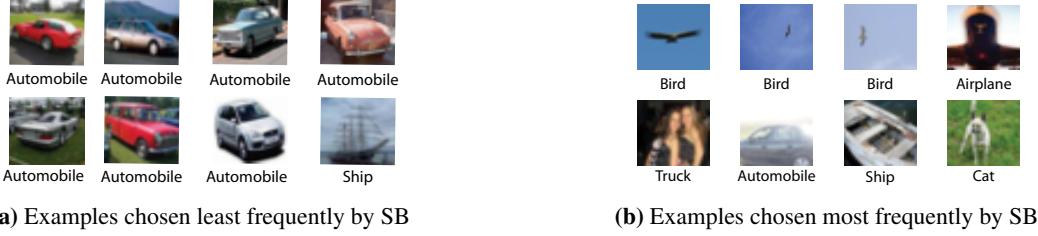


Figure 2: Example images from CIFAR10

Importance sampling and **curriculum learning** are common techniques for generalizing DNNs. Differing philosophies motivate these approaches: **supplying easy or canonical examples early in training as in self-paced learning** (Bengio et al., 2009; Kumar et al., 2010), **emphasizing rare or difficult examples to accelerate learning, or avoiding overfitting** (Alain et al., 2015; Jiang et al., 2015; Katharopoulos & Fleuret, 2017), **targeting marginal examples that the network oscillates between classifying correctly and incorrectly** (Chang et al., 2017), or taking a black-box, data-driven, approach (Jiang et al., 2018; Ma et al., 2017; Ren et al., 2018). These works **improve target accuracy on image classification tasks, and datasets with high label error** (Jiang et al., 2018; Ren et al., 2018). These techniques, however, do **not target and analyze training speedup** (Bengio et al., 2009; Chang et al., 2017; Katharopoulos & Fleuret, 2017; Kumar et al., 2010), often **adding overhead** to the training process by, e.g., **training an additional DNN** (Katharopoulos & Fleuret, 2017; Jiang et al., 2018; Zhang et al., 2019) or **performing extra training passes** on a separate validation set (Ren et al., 2018). For instance, Zhang et al. (2019) requires running an additional DNN asynchronously on separate hardware to speed up training.

### 3 LOSS-BASED SAMPLING WITH SELECTIVE-BACKPROP

#### 3.1 Background

Selective-Backprop can be applied to standard mini-batch stochastic gradient descent (SGD), and is compatible with variants such as AdaGrad, RMSprop, and Adam that differ only in learning rate scheduling. The **goal of SGD** is to find parameters  $\mathbf{w}^*$  that **minimize the sum of the losses  $\mathcal{L}$**  for a model  $f(\mathbf{w})$  with  $d$  parameters over all points (indexed by  $i$ ) in a dataset  $\mathcal{D}$  consisting of  $n$  examples  $(\mathbf{x}_i, y_i)$ .

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^n \mathcal{L}(f_{\mathbf{w}}(\mathbf{x}_i), y_i)$$

SGD proceeds in a number of iterations, at each step selecting a single example  $i$  and updating the weights by subtracting the gradient of the loss multiplied by a step-size

parameter  $\eta$ .

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{L}(f_{\mathbf{w}_t}(\mathbf{x}_i), y_i)$$

In minibatch gradient descent, at **each step**, one selects a **subset of examples  $\mathcal{M}_t$** , often by **sampling from  $\mathcal{D}$  at random without replacement**, traversing the full training set once per epoch, applying the update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \sum_{x_i, y_i \in \mathcal{M}_t} \nabla_{\mathbf{w}} \mathcal{L}(f_{\mathbf{w}_t}(\mathbf{x}_i), y_i)$$

We refer to this approach as **Traditional**.

---

#### Algorithm 1 Selective-Backprop training cycle.

---

```

function Train(data, bSize)
    batch_bp ← []
    for batch_fp in data.getBatches(bSize) do
        losses ← n.Forward(batch_fp)
        for exp, loss in zip(examples, losses) do
            prob ← sb.CalcProb(loss)
            chosen ← choose(prob)
            if chosen then
                batch_bp.append(exp, loss)
            end if
            if batch_bp.size == bSize then
                n.Backward(batch_bp)
                batch_bp ← []
            end if
        end for
    end for
end for

```

---



---

#### Algorithm 2 Selective-Backprop's probability calculation.

---

```

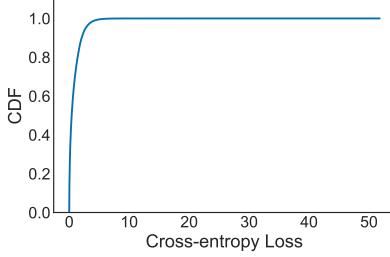
hist ← deque(histSize)
function CalculateProbability(loss)
    hist.add(loss)
    perc ← percentile(hist, loss)
    prob ← percβ
    return prob

```

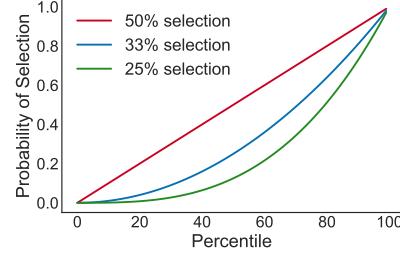
---

#### 3.2 Selective-Backprop

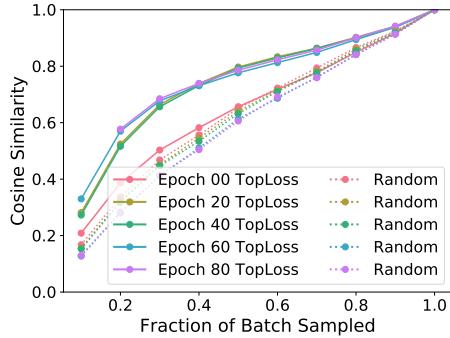
Selective-Backprop also traverses the training set once per epoch, but, like other selection-based acceleration tech-



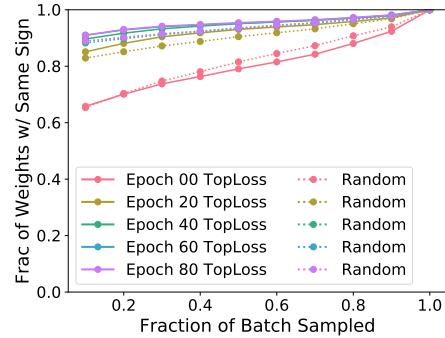
**Figure 3:** Snapshot of a CDF of cross-entropy losses when training MobilenetV2 with CIFAR10



**Figure 4:** Selective-Backprop calculates the probability of selection using  $\mathcal{L}(d)$  as a percentile of the  $R$  most recent losses



(a) Cosine similarity



(b) Fraction of gradient components with same sign

**Figure 5:** Similarity between gradients calculated on the original batch and subsampled batches when training MobilenetV2 on CIFAR10. After the first epoch, high-loss examples are more similar than random subsampling by both cosine similarity and fraction of weights with same sign.

niques, it generates batches using a non-uniform selection criteria designed to require fewer backward pass calculations to reach a given loss. To construct batches, Selective-Backprop selects each example with a probability that is a function of its current loss as determined by a forward pass through the network:  $\mathcal{P}(\mathcal{L}(f_w(x_i), y_i))$ .

In each epoch, Selective-Backprop randomly shuffles the training examples  $\mathcal{D}$  and iterates over them in the standard fashion. However, for each example  $i$ , after computing a forward pass to obtain its loss  $\mathcal{L}(f_w(x_i), y_i)$ , Selective-Backprop then decides whether to include the example for a gradient update by selecting it with probability  $\mathcal{P}(\mathcal{L})$  that is a function of the current loss. Selecting a sufficient number of examples for a full batch ( $\mathcal{M}_t$ ) for a gradient update typically requires forward pass calculations on more than  $\mathcal{M}_t$  examples. After collecting a full batch, SB updates the network using gradients calculated based on this batch. Alg. 1 details this algorithm.

Setting the selection probability to 1 for all examples expresses standard minibatch SGD. For Selective-Backprop, we develop an intuitive heuristic whereby examples with higher loss are more frequently included in updates (Fig-

ure 2b), while those with the lower losses (Figure 2a) are included less frequently. Our experiments show that suppressing gradient updates for low-loss examples has surprisingly little impact on the updates. For example, we empirically find that the sign of over 80% of gradient weights is maintained, even when subsampling only 10% of the data with the highest losses (Fig. 5b). Since recent research has demonstrated that the sign of the gradient alone is sufficient for efficient learning (Bernstein et al., 2018), this bodes well for our method. Moreover, gradients calculated with only the highest loss examples maintain higher cosine similarity to those calculated with all examples as compared to randomly subsampling examples in a batch (Fig. 5a).

Alg. 2 details our heuristic for setting  $\mathcal{P}(\mathcal{L})$ . We set  $\mathcal{P}(\mathcal{L})$  to be a monotonically increasing function of the CDF of losses across the example set. In Figure 3, we show an example of historical losses snapshotted during training. Because recomputing the complete CDF after each update is not practical, we approximate the current CDF using a running tally of the losses of the last  $R$  examples, denoted by  $CDF_R$ :

$$\mathcal{P}(\mathcal{L}(f_w(x_i), y_i)) = [CDF_R(\mathcal{L}(f_w(x_i), y_i))]^\beta, \quad (1)$$

where  $\beta > 0$  is a constant that determines Selective-Backprop’s level of selectivity and thus allows us to modulate the bias towards high-loss examples where larger values produce greater selectivity (Figure 4). We include a sensitivity analysis of  $\beta$  in Section 5.

### 3.3 Reducing selection overhead

Selective-Backprop accelerates training by reducing the number of backward passes needed to reach given levels of loss. In our experiments (Section 5), we find that after reducing backward passes with Selective-Backprop, the largest remaining fraction of training time is the full (original) complement of forward passes used to select the Selective-Backprop batches. We distinguish these forward passes from the forward passes used for training by referring to them as “selection passes” in the rest of the paper. This section describes four approaches to reducing the time spent in selection passes of Selective-Backprop training, thereby further reducing overall training time.

**Re-using previous losses.** Selective-Backprop’s selection pass uses the latest model parameters to compute up-to-date losses for all training examples considered. We define and evaluate a Selective-Backprop variant called Stale-SB that executes selection passes every  $n^{\text{th}}$  epoch. The subsequent  $(n - 1)$  epochs reuse the losses computed by the previous selection pass to create the backprop batch. The losses are reused in the following epoch(s), but only for batch formation. Intuitively, if an example is deemed important in a given selection pass, it will also have a high probability of being selected in the next  $(n - 1)$  epochs. Stale-SB with  $n = 1$  is Selective-Backprop. We evaluate Stale-SB in Section 5 and find that, typically, it reduces selection pass cost significantly without impacting final accuracy.

**Using predicted losses.** Rather than simply re-using the loss from an earlier epoch for selecting examples, one could construct a Selective-Backprop variant that predicts the losses of examples using historical loss values. To make the problem easier, instead of predicting the loss directly, one could predict whether the loss is high enough to cross the threshold for selection. We evaluated various prediction approaches, including tracking a exponentially weighted average of historical losses and using historical losses to train a Gaussian process predictor. None outperformed the simpler Stale-SB approach.

**Pipelining loss computation.** Given multiple computation engines, one could construct a Selective-Backprop variant that selects examples for batch  $N + 1$  on a separate engine while training with batch  $N$  is ongoing. Such an approach would require using losses computed from stale versions of the model, but could mask nearly all training delays for selection and could do so without the “history size” concerns that would arise for the above approaches when training

with giant or continuous datasets. Running a separate model for loss computation would, however, introduce a new overhead of occasionally syncing the selection model to reflect changes to the training model.\* This introduces a new trade-off between frequency of syncing the selection model and the amount of staleness introduced to the selection process.

Although one could use equivalent GPUs for such pipelining, it is unclear that this would be better than data-parallel training. Rather, we think the natural application of the pipelining approach would be in combination with the inference accelerators discussed next—that is, a low-cost inference accelerator could be used to compute losses for example selection, and then a powerful compute engine could be used for training on batches of selected examples.

**Inference accelerators.** For general-purpose hardware and training frameworks, the cost of the backward pass is approximately twice the forward pass cost. This is because during the forward pass, we calculate one convolution per convolutional layer, whereas in the backward pass we perform two: one convolution to calculate the gradients w.r.t the input data and another w.r.t to the layer weights (Ben-Nun & Hoefer, 2018). But, a variety of inference acceleration approaches, such as reduced precision or quantization, may enable specialized hardware accelerators to run forward passes  $\approx 10x$  faster than a backward pass on a modern GPU (Jouppi et al., 2017). Since SB selects examples by running a forward pass, it can use such accelerators. Although aggressive forward-pass acceleration can affect the outcome of training, use of inference acceleration for Selective-Backprop’s selections may not have the same negative consequences. We leave exploration of this approach to reducing selection time to future work, but include it in this list for completeness.

## 4 IMPLEMENTATION DETAILS

We built prototypes of SB for PyTorch 0.4.1 and Keras 2.2.4; the Section 5 evaluation is based on the former.

To add SB into existing training code, we introduce a mathematically simple probabilistic filtering step to training, which down-selects examples used for updates. Filtering starts by calculating the loss for each example using a forward pass. SB adds the loss to  $\text{CDF}_R$  (implemented using a bounded queue), and calculates what percentile of losses it represents. Using this percentile, SB calculates the selection probability  $\mathcal{P}$ , and probabilistically adds this example to the minibatch for training. We measure the overhead introduced by SB’s selection step, excluding time spent in selection passes, to  $\approx 3\%$  of overall training time.

Selective-Backprop’s lightweight filtering step is simple to

\*Using modern deep learning frameworks such as PyTorch, we found that copying a new model and moving it to a second device can take up to a minute.

implement and requires few changes to existing training code. In traditional setups, data is formed into minibatches for training. In SB, data is formed into selection minibatches and fed into SB’s filtering mechanism. SB performs forward passes of selection minibatches (“selection passes”), forms a training minibatch of selected data examples, and passes it to the original training code. Therefore, the training code can be agnostic to SB’s filtering mechanism, allowing SB to work in tandem with any training optimizer (e.g., SGD, Adam) and common optimizations such as batch normalization, data augmentation, dropout, and cutout.

**Future implementation optimizations.** Our SB implementation minimizes changes to existing code, and some obvious potential optimizations are not currently incorporated. For instance, in our implementation, two forward passes are performed for each selected example: one for selection and one for training. Unless selection passes are accelerated using reduced precision or quantization, which is not the case in our implementation, a more optimized SB implementation could cache the activations obtained from the selection passes to avoid doing extra forward passes for training, and thus eliminate the time spent in “Forwards (training)” for SB shown in Figure 1. Another optimization would use a minibatch size for selection that is larger than that of training, to reduce the number of selection passes needed to populate a training minibatch.

## 5 EVALUATION

We evaluate Selective-Backprop’s effect on training with modern image classification models using CIFAR10, CIFAR100, and SVHN. The results show that, compared to traditional training and a state-of-the-art importance sampling approach (Katharopoulos & Fleuret, 2018), SB reduces wall-clock time needed to reach a range of target error rates by up to 3.5x (Section 5.2). We show that by reducing the time spent in example selection, one can further accelerate training by on average 26% (Section 5.3). Additional analyses show the importance of individual SB characteristics, including selection of high-loss examples and robustness to label error (Section 5.4). Throughout the evaluation, we also show that the speedup achieved by a SB depends the learning rate schedule, sampling selectivity, and target error. Section 5.5 shows that, across a sweep of configurations, the majority of Pareto-optimal trade-off points come from Selective-Backprop and Stale-SB. We also provide a sensitivity analysis for SB in Section 5.4 and the results from two additional learning rate schedules in the appendix.

### 5.1 Experimental Setup

We train Wide Resnet, ResNet18, DenseNet, and MobileNetV2 (Zagoruyko & Komodakis, 2016; He et al., 2016;

Huang et al., 2017; Sandler et al., 2018) using SB, Traditional as described in Section 3, and our implementation of Kath18 (Katharopoulos & Fleuret, 2018) with variable starting, no bias reweighting, and using loss as the importance score criteria. We tune the selectivity of SB and Kath18 individually for each dataset. In our evaluation, we present the results of training Wide Resnet. To train Wide Resnet, we use the training setup specified by (DeVries & Taylor, 2017), which includes standard optimizations including cutout, batch normalization and data augmentation. We observe similar trends when using ResNet18, DenseNet, and MobileNetV2, and present those results in the appendix. In each case, we do not retune existing hyperparameters. We report results using the default batch size of 128 but confirm that the trends remain when using batch size 64.

**CIFAR10.** The CIFAR10/100 datasets (Krizhevsky, 2009) contain 50,000 training images and 10,000 test images, divided into 10 and 100 classes, respectively. Each example is a 32x32 pixel image. We use  $batch\_size = 128$  and cutout of length 16. We use an SGD optimizer with  $decay = 0.0005$ . We train with two learning rate schedules. In the first schedule, we start with  $lr = 0.1$  and decay by 5x at 60, 120, and 160 epochs. In the second schedule, we decay at 48, 96, and 128 epochs. We use 33% selectivity for SB, Stale-SB, and Kath18. Training ends after 12 hours.

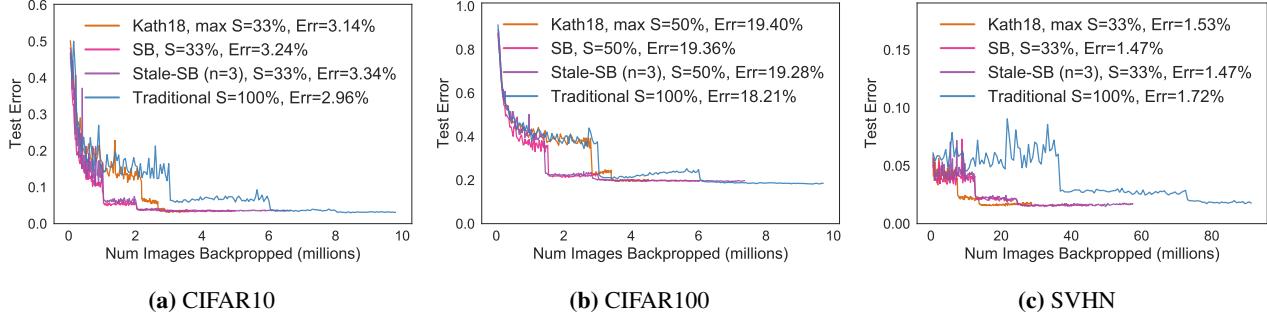
**CIFAR100.** We train on CIFAR100 using the setup specified by DeVries & Taylor (2017). We use  $batch\_size = 128$  and cutout of length 8. We train with two learning rate schedules. First, we start with  $lr = 0.1$  and decay by 10x at 60 and 120. In the second learning rate schedule, we decay at 48 and 96 epochs. We use 50% selectivity for SB, Stale-SB, and Kath18. Training ends after 12 hours.

**SVHN.** SVHN has 604,388 training examples and 26,032 testing examples of digits taken from Street View images (Netzer et al., 2011). For the first schedule, we initialize the learning rate to 0.1 and decay to 0.01 and 0.001 at epochs 60 and 120. For the second, we decay at 48 and 96. We use  $batch\_size = 128$  and cutout of length 20. We use 25% selectivity for SB and Stale-SB, and 33% selectivity for Kath18. Training ends after 96 hours.

**Hardware.** We train CIFAR10 and CIFAR100 on servers equipped with 16-core Intel Xeon CPUs, 64 GB RAM and NVIDIA TitanX GPUs. We train SVHN on servers with four 16-core AMD Opteron CPUs, 128 GB RAM, and NVIDIA Tesla K20c GPUs.

### 5.2 Selective-Backprop speeds up training

**SB reduces training iterations to target error.** SB probabilistically skips the backward passes of examples with low loss in order to learn more per example. Figure 6 shows that SB reaches final or non-final target error rates with fewer



**Figure 6:** SB reduces training iterations to target error.  $S$  is the selectivity used, and  $Err$  is the final test error reached.

training iterations (updates to the network). This can be seen by comparing the x-axis points at which lines for each approach reach a particular y-axis value. Note that we plot different y-axes for different datasets. Although the savings depends on the specific target test error rate chosen, one way to visualize the overall speedup across different target accuracy is by comparing the area under the three curves. SB reaches nearly every test error value with significantly fewer training iterations.

**SB reduces wall-clock time to target error.** Figure 7 shows error rate as a function of wall-clock time. SB speeds up time to target error rates by reducing backward passes, without optimizations to reduce selection time discussed in Section 5.3. Table 1 shows that for CIFAR10, SB reaches within 10%, 20%, and 40% of Traditional’s final error rate 1.2–1.5x faster. For SVHN, SB provides a 3.4–5x speedup to reach 1.8%, 2.1%, and 2.4% error.

Intuitively, SB is most effective on datasets with many redundant examples or examples that are easy to learn. CIFAR100 is a more challenging dataset for sampling as there are fewer examples per class and therefore likely less redundancy. Despite this, SB reaches within 20% and 40% of the Traditional’s final error rate 20% faster. However, it sacrifices a small amount of final accuracy for these speedups and does not reach within 10% of Traditional’s final error rate in the allotted training time. Kath18 also accelerates training over Traditional by 0.8–3.4x. Similarly to SB, it is most effective on SVHN and least effective on CIFAR100, even leading to a small slowdown to certain target error rates. SB provides a speedup over Kath18 of 1.02–1.8x.

**Selective-Backprop performs better on challenging examples.** SB converges faster than Traditional by outperforming Traditional on challenging examples. Figure 8 shows an inverse CDF of the network’s confidence in each ground truth label of the test set; the data represents a snapshot in time after training SB or Traditional for ten epochs. For each percentile, we plot the target confidence on the y-axis (e.g., the 20<sup>th</sup> percentile of target confidences for SB is 55%). The network’s classification is the class with

the Top-1 confidence (using argmax). Therefore, we cannot infer the classification accuracy of an example solely from its target confidence (if the target confidence is  $\leq 50\%$ ). In Figure 9, we also plot the accuracy of each percentile of examples. Generally, examples at lower percentiles are harder for the network to accurately classify. Using SB, the network has higher confidence and accuracy in these lower percentiles. For instance, among the examples at the 20th percentile of target confidences, 29% of these examples are classified correctly using SB while only 3% are classified correctly by Traditional. While this comes at the cost of confidence in higher percentile examples, test accuracy is not sacrificed. In fact, SB is able to generalize better across all examples of all difficulty levels.

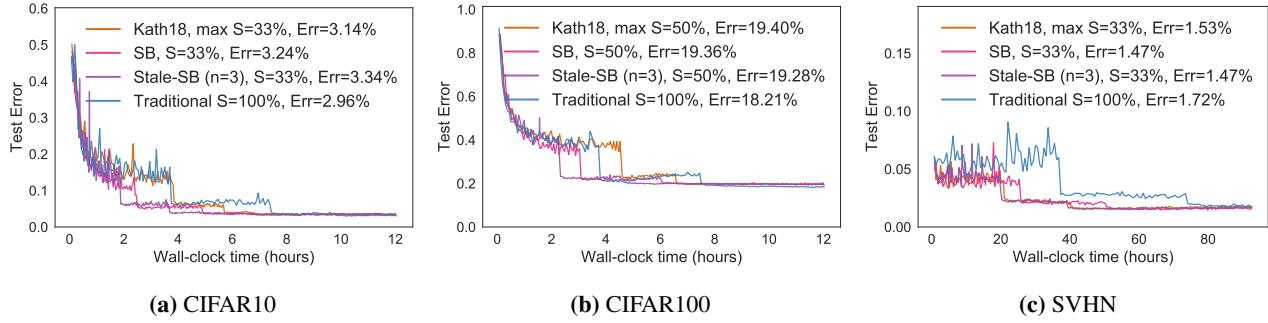
### 5.3 Reducing selection times further speeds training

In Figure 10, we see that SB reduces total time to target error rates compared to Traditional by reducing the time spent in the backward pass. In Figure 10a and Figure 10b, both Traditional and SB take the same number of epochs to reach the target error rate. However, SB performs more overall forward passes. As described in Section 4, this is because we perform one selection forward pass for each candidate example, plus one training forward pass for each selected example. The “Other” bar shown for each run includes per-run overheads (e.g., loading the dataset into memory and the network onto the GPU) and per-epoch overheads (e.g., evaluating test accuracy).

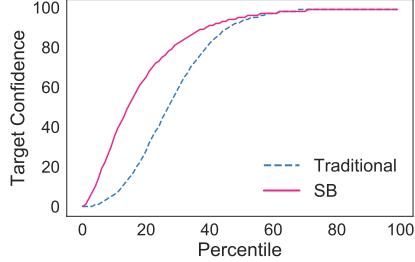
**Using stale losses to reduce selection passes.** After SB’s reduction of backward passes performed, over half of the remaining training time is spent on forward passes. We evaluate Stale-SB, which uses the losses of forward passes from previous epochs to perform selection. With Stale-SB, we run fewer selection passes, running them only every  $n = 2$  or  $n = 3$  epochs. That is, if  $n = 2$ , an example that incurs a high loss has a high chance of being trained on in the next two epochs, instead of just one. In Figure 10, we see that Stale-SB with  $n = 3$  reduces the time spent performing selection passes by two-thirds, thereby further

Dataset	Strategy	Final error of Traditional	Speedup to final error $\times 1.1$	Speedup to final error $\times 1.2$	Speedup to final error $\times 1.4$
CIFAR10	SB	2.96%	1.4x	1.2x	1.5x
CIFAR10	Stale-SB	2.96%	—	1.5x	2.0x
CIFAR10	Kath18	2.96%	1.4x	1.1x	1.3x
CIFAR100	SB	18.21%	1.2x	1.2x	1.2x
CIFAR100	Stale-SB	18.21%	1.5x	1.0x	1.6x
CIFAR100	Kath18	18.21%	1.1x	0.8x	0.8x
SVHN	SB	1.72%	3.4x	3.4x	3.5x
SVHN	Stale-SB	1.72%	4.3x	4.9x	5.0x
SVHN	Kath18	1.72%	1.9x	2.8x	3.4x

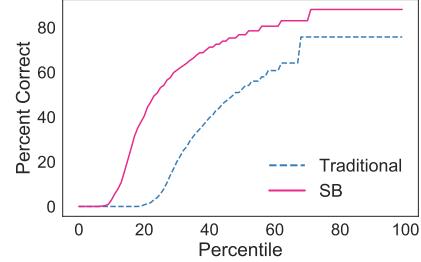
**Table 1:** Speedup achieved by SB and Kath18 over Traditional



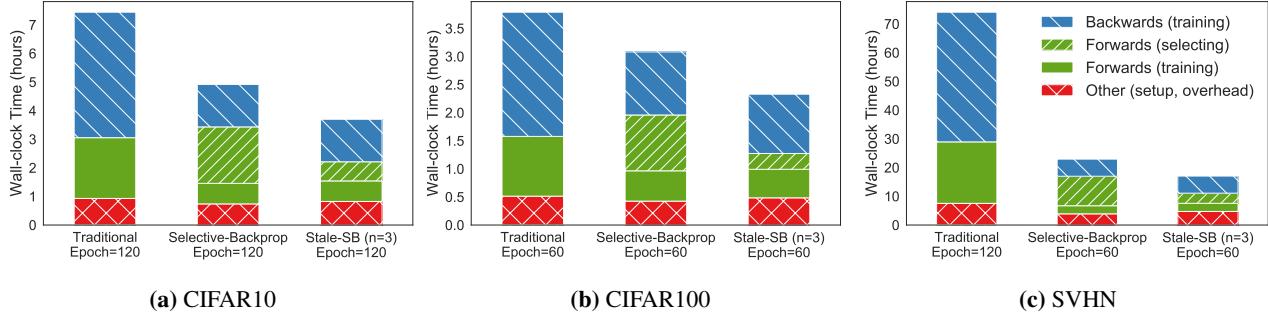
**Figure 7:** SB reduces wall-clock time to target error.  $S$  is the selectivity used, and  $Err$  is the final test error reached.



**Figure 8:** SB has higher confidence in harder examples with almost no cost of confidence in easy examples.



**Figure 9:** SB increases accuracy of harder examples, without sacrificing accuracy of easy examples.



**Figure 10:** SB reduces time spent in the backward pass in order to speed up time to the target error rate (in this case, 1.4x of Traditional's final error rate). Stale-SB further accelerates training by reducing the time spent performing selection passes.

reducing the total wall-clock time. In Figure 11, we see that the reduced number of total forward passes in Stale-SB has little effect on final error. Stale-SB’s ability to reduce selection passes while maintaining test accuracy leads to the end-to-end speedups shown in Table 1. On average, Stale-SB with  $n = 3$  reaches target error rates 26% faster than SB. With  $n = 3$ , we believe Stale-SB captures most of the benefits of reducing selection passes, though we have not yet experimented with values of  $n > 3$ .

#### 5.4 Selective-Backprop sensitivity analysis

**SB is robust to modest amounts of label error.** One potential downside of SB is that it could increase susceptibility to noisy labels. However, we show that on SVHN, a dataset known to include label error (Papernot & McDaniel, 2018), SB still converges faster than Traditional to almost all target error rates. We also evaluate SB on CIFAR10 with manually corrupted labels. Following the UniformFlip approach in (Ren et al., 2018), we randomly flip 1% (500 examples), 10% (5000 examples) and 20% (10000 examples).

Fig 12 shows that SB accelerates training for all three settings. With 1% and 10% of examples corrupted, SB reaches a comparable final test accuracy. With 20% corruption, SB overfits to the incorrect labels and increases the final test error. So, while SB is robust to modest amounts of label error, it is most effective on relatively clean, validated datasets.

**Higher selectivity accelerates training, but increases final error.** Tuning  $\beta$  in Equation 1 changes SB’s selectivity. In Figure 13, we see that increasing SB’s selectivity, focusing more on harder examples, increases the speed of learning but can cause result in higher final error. For CIFAR10, SB reaches within 0.92% of Traditional’s final error rate with 20% selectivity. For CIFAR100, it reaches within 2.54% of the final error rate with 25% selectivity. As with other hyperparameters, the best selectivity depends on the target error and dataset. Overall, we observe that SB speeds up training with a range (20–65%) for selectivity.

**SB using additional learning rate schedules.** We train SB using the provided learning rate schedule in (DeVries & Taylor, 2017) which reproduces state-of-the-art accuracies on CIFAR10, CIFAR100, and SVHN for Wide Resnet with Cutout. We also train using a static learning rate schedule to adjust for confounding factors, as well as an accelerated learning rate schedule. In both cases, we see the same trends as with the initial learning rate. We include the configurations in Section 5.5 and the training curves in the appendix.

#### 5.5 Putting it all together

The optimal training setup to reach a certain target error rate depends on a variety of factors. In the previous sections, we compared Traditional, SB, Stale-SB, and Kath18 using a

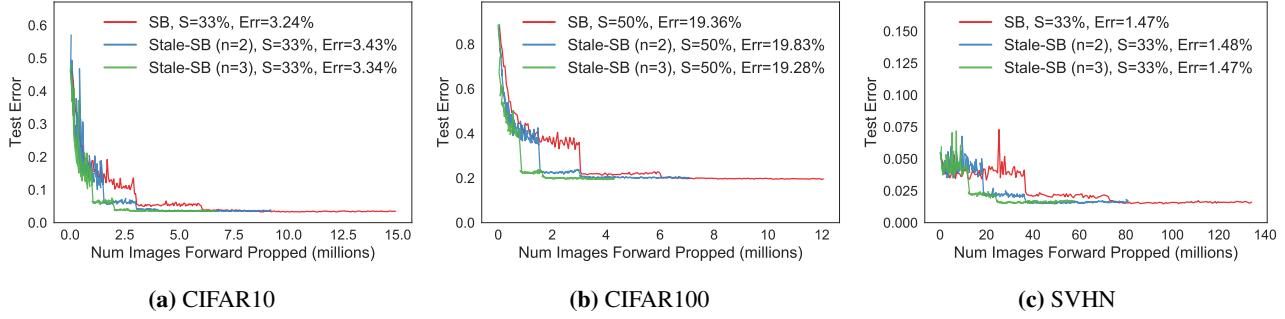
variety of different configurations. In Figure 14, we plot the wall-clock time needed to reach a range of target error rates for all four strategies, each trained with two learning rate schedules and run with different selectivities. A small subset of configurations make up the Pareto frontier, which represent the best strategy for a given target error rate. Points on the Pareto frontier are colored in bold whereas suboptimal points are shown with transparency.

**SB provides the majority of Pareto-optimal configurations.** As an approximate signal for robustness of our strategy, we calculate the fraction of Pareto points provided by SB and Stale-SB, Kath18 and Traditional. For a majority of training time budgets, SB gives the lowest error rates. For CIFAR10, CIFAR100, and SVHN, SB and its optimized variant Stale-SB account for 72%, 47% and 80% of the Pareto-optimal choices, respectively. The exception is cases with very large training time budgets, where Traditional reaches lower final error rates than SB. Overall, Traditional accounts for 10%, 43% and 6% of Pareto points in CIFAR10, CIFAR100 and SVHN, respectively. As shown in Table 1, SB is also faster than Kath18, a state-of-the-art importance sampling technique for speeding up training, while achieving the same final error rate. Kath18 provides 10%, 8% and 14% of Pareto-optimal points.

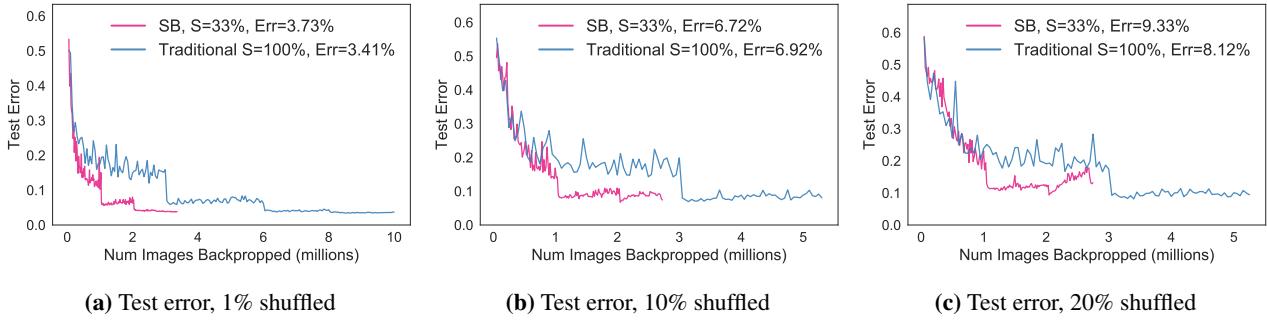
**Practicality.** Selective-Backprop reduces training iterations and wall-clock time needed to achieve a target error rate with little programmer effort. We evaluated Selective-Backprop with a diverse set of network architectures and datasets. In each case, we did not retune initial hyperparameters from canonical setups. Most of these training setups included traditional accuracy-boosting techniques, including data augmentation, cutout, dropout, and batch normalization. Selective-Backprop still improved training atop these existing optimizations. Selective-Backprop is also mathematically lightweight and simple to add to code.

## 6 CONCLUSION

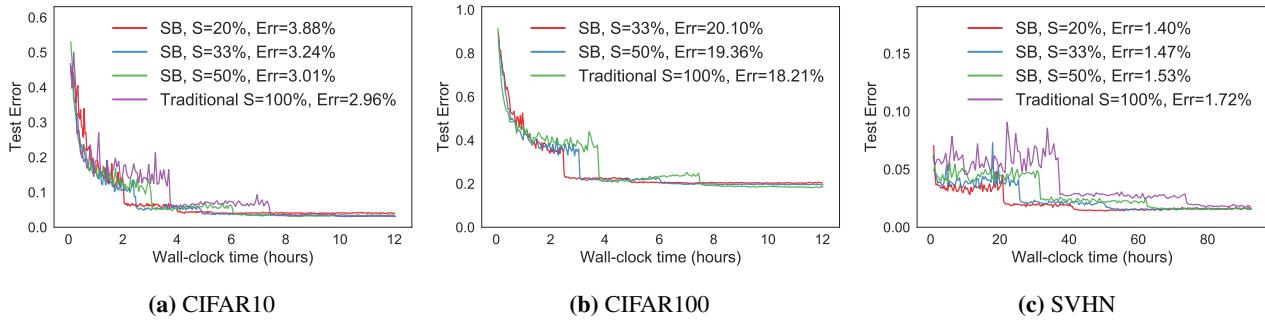
Selective-Backprop accelerates training of neural networks. The key idea is to skip the backward pass for training examples for which the forward-pass loss function indicates little value. Selective-Backprop lets the current state of the network dictate selectivity for each example, and is lightweight, scalable, and effective. Experiments on several datasets and networks show that Selective-Backprop converges to target error rates up to 3.5x faster than with standard SGD and 1.02–1.8x faster than the state-of-the-art sampling approach introduced in (Katharopoulos & Fleuret, 2018). Determining selectivity with stale loss information further accelerates training by 26%. Selective-Backprop is also simple to add to code. An open source implementations of Selective-Backprop is available at <https://bit.ly>SelectiveBackpropAnon>.



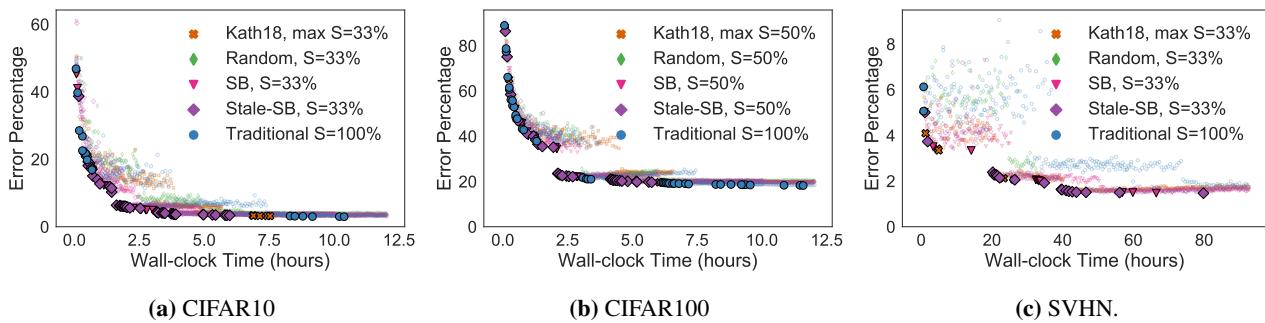
**Figure 11:** Increasing loss staleness reduces number of forward passes with little loss in accuracy.



**Figure 12:** SB reaches similar test error rates compared to Traditional with 1% and 10% shuffled labels.



**Figure 13:** SB accelerates training for a range of selectivities. Higher selectivity gives faster training but can increase error.



**Figure 14:** Pareto-optimal points for training time vs error trade-off are opaque and filled. SB and Stale-SB offer the majority of Pareto-optimal options for trading off training time and accuracy.

## REFERENCES

- Alain, G., Lamb, A., Sankar, C., Courville, A., and Bengio, Y. Variance reduction in sgd by distributed importance sampling. *arXiv preprint*, arXiv:1511.06481, Nov 2015.
- Ben-Nun, T. and Hoefer, T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys*, 02 2018.
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.
- Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. signSGD: Compressed Optimisation for Non-Convex Problems. In *International Conference on Machine Learning (ICML)*, 2018.
- Chang, H.-S., Learned-Miller, E. G., and McCallum, A. Active Bias: Training a more accurate neural network by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, 2017.
- Chintala, S. Convnet-Benchmarks. <https://github.com/soumith/convnet-benchmarks>.
- DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
- Gao, J., Jagadish, H. V., and Ooi, B. C. Active Sampler: Light-weight accelerator for complex data analytics at scale. *arXiv preprint*, arXiv:1512.03880, 2015.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- Hinton, G. E. To recognize shapes, first learn to generate images. In *Computational Neuroscience: Theoretical Insights into Brain Function*, Progress in Brain Research. Elsevier, 2007.
- Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.
- Jiang, L., Meng, D., Zhao, Q., Shan, S., and Hauptmann, A. G. Self-paced curriculum learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2015.
- Jiang, L., Zhou, Z., Leung, T., Li, L.-J., and Fei-Fei, L. MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning (ICML)*, 2018.
- Johnson, T. B. and Guestrin, C. Training deep models faster with robust, approximate importance sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-I., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017.
- Katharopoulos, A. and Fleuret, F. Biased importance sampling for deep neural network training. *CoRR*, abs/1706.00043, 2017.
- Katharopoulos, A. and Fleuret, F. Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning (ICML)*, 2018.
- Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.
- Kuang, L. Train CIFAR10 with PyTorch. <https://github.com/kuangliu/pytorch-cifar>, Jan 2019.
- Kumar, M. P., Packer, B., and Koller, D. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2010.
- Lin, T.-Y., Goyal, P., Girshick, R. B., He, K., and Dollár, P. Focal loss for dense object detection. In *International Conference on Computer Vision (ICCV)*, 2017.
- Loshchilov, I. and Hutter, F. Online batch selection for faster training of neural networks. *International Conference on Learning Representations (ICLR)*, 2015.
- Ma, F., Meng, D., Xie, Q., Li, Z., and Dong, X. Self-paced co-training. In *International Conference on Machine Learning (ICML)*, 2017.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Papernot, N. and McDaniel, P. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

Ren, M., Zeng, W., Yang, B., and Urtasun, R. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning (ICML)*, 2018.

Rosasco, L., De Vito, E., Caponnetto, A., Piana, M., and Verri, A. Are loss functions all the same? *Neural Comput.*, 2004.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2016.

Shrivastava, A., Gupta, A., and Girshick, R. Training region-based object detectors with online hard example mining. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, J., Yu, H.-F., and Dhillon, I. S. Autoassist: A framework to accelerate training of deep neural networks. *ArXiv*, abs/1905.03381, 2019.

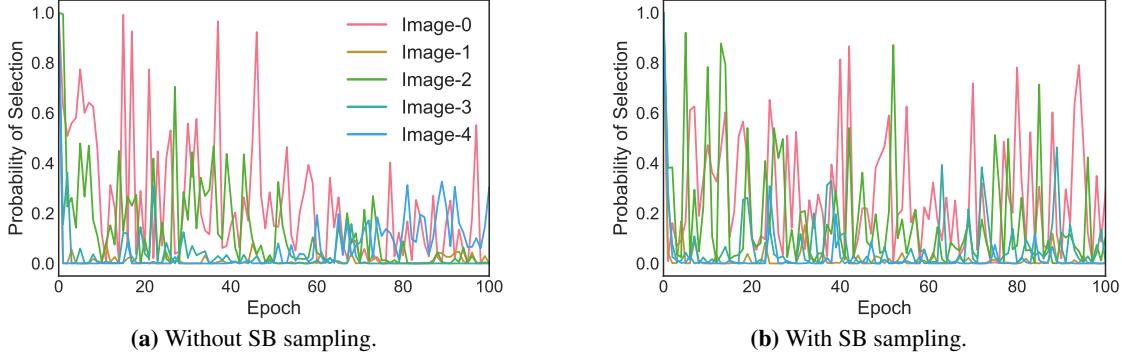
## A SUPPLEMENTARY MATERIALS

### A.1 Selective-Backprop code

Our implementation of Selective-Backprop can be found at <https://bit.ly/SelectiveBackpropAnon>. The specific training set up used in this paper is available at <http://bit.ly/SBCutoutAnon>, which we modified to use our implementation as a submodule.

### A.2 Variance of relative losses over time

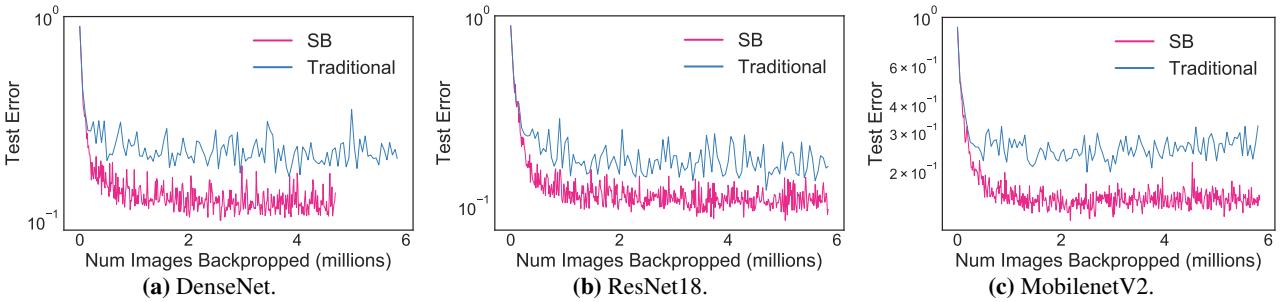
Hinton (2007) warns that when sampling, easy examples are likely to develop large gradients while being ignored. We show this effect on CIFAR10. In typical training, the probability of selection has a natural variance, but increases once sampling is introduced. One benefit of using a look-once approach for determining example importance is that Selective-Backprop always uses the up-to-date state of the network, which is in constant flux during training.



**Figure 15:** Select probabilities of five examples when training MobilenetV2 on CIFAR10. Each line represents one image. Likelihood of selection fluctuates more when sampling is introduced.

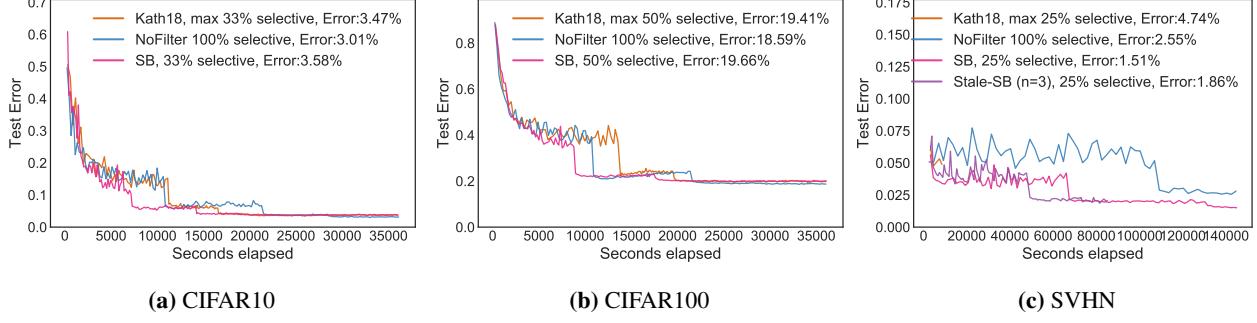
### A.3 Sensitivity analyses

**Network architecture.** We train CIFAR10 using three architectures, ResNet18, DenseNet, and MobileNetV2 (He et al., 2016; Huang et al., 2017; Sandler et al., 2018) We train CIFAR10 using the training setup from pytorch-cifar (Kuang, 2019). The learning rate is set at 0.1.



**Figure 16:** Training on CIFAR10 using three different network architectures.

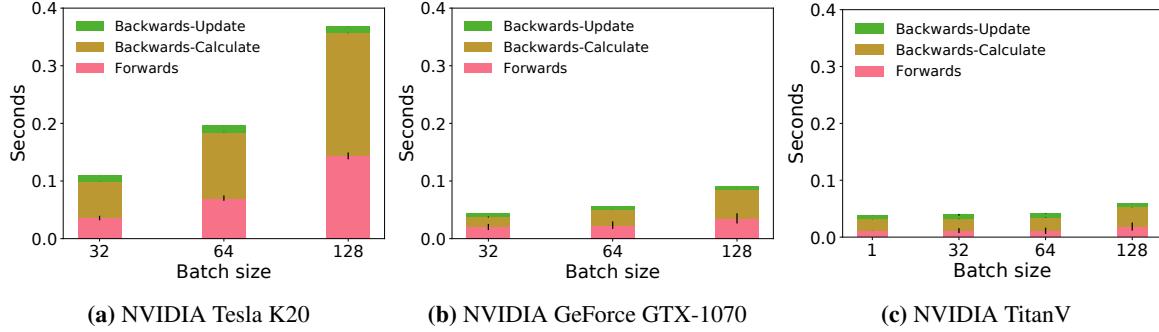
**Additional learning rate schedule.** We train CIFAR10, CIFAR100 and SVHN on an additional, accelerated learning rate schedule and observe the same trends as in Section 5. For CIFAR10, we start with  $lr = 0.1$  and decay by 5x at 48, 96, and 128 epochs. For CIFAR100, we start with  $lr = 0.1$  and decay by 10x at 48 and 96 epochs. For SVHN we initialize the learning rate to 0.1 and decay by 10x at epochs 60 and 120.



**Figure 17:** SB reduces wall-clock time to target error with an accelerated learning rate schedule.

#### A.4 Asymmetry between cost of backward and forward pass

In Figure 18, we confirm that on both a variety of modern GPUs, the backward pass takes up to 2.5x as long as the forward pass. In Section 5, we run experiments on the K20 and the TitanV with a batch size of 128.



**Figure 18:** Breakdown of processing time per batch while training MobileNetV2 with Traditional.