A Multigrid Method for Efficiently Training Video Models

Chao-Yuan Wu[1,2]   Ross Girshick[2]   Kaiming He[2]
Christoph Feichtenhofer[2]   Philipp Krähenbühl[1]

[1]The University of Texas at Austin   [2]Facebook AI Research (FAIR)

Abstract

Training competitive deep video models is an order of magnitude slower than training their counterpart image models. Slow training causes long research cycles, which hinders progress in video understanding research. Following standard practice for training image models, video model training has used a fixed mini-batch shape: a specific number of clips, frames, and spatial size. However, what is the optimal shape? High resolution models perform well, but train slowly. Low resolution models train faster, but are less accurate. Inspired by multigrid methods in numerical optimization, we propose to use variable mini-batch shapes with different spatial-temporal resolutions that are varied according to a schedule. The different shapes arise from resampling the training data on multiple sampling grids. Training is accelerated by scaling up the mini-batch size and learning rate when shrinking the other dimensions. We empirically demonstrate a general and robust grid schedule that yields a significant out-of-the-box training speedup without a loss in accuracy for different models (I3D, non-local, SlowFast), datasets (Kinetics, Something-Something, Charades), and training settings (with and without pre-training, 128 GPUs or 1 GPU). As an illustrative example, the proposed multigrid method trains a ResNet-50 SlowFast network 4.5× faster (wall-clock time, same hardware) while also improving accuracy (+0.8% absolute) on Kinetics-400 compared to baseline training. Code is available online.[1]

1. Introduction

Training deep networks (CNNs [27]) on video is more computationally intensive than training 2D CNN image models, potentially by an order of magnitude. Long training time slows progress in video understanding research, hinders scaling out to real-world data sources, and consumes significant amounts of energy and hardware. Is this slow training unavoidable, or might there be video-specific optimization strategies that can accelerate training?

3D CNN video models are trained using mini-batch optimization methods (*e.g.*, SGD) that process one mini-batch
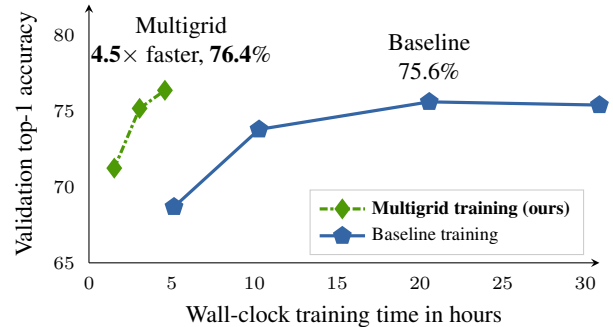

Figure 1. **Training time *vs*. top-1 accuracy on Kinetics-400** with a ResNet-50 SlowFast network. Each point corresponds to a model trained for a specific number of epochs. **Multigrid training**, the method developed in this paper, obtains a significantly better trade-off than baseline training. For example, under default settings, multigrid training is **4.5×** faster while achieving higher (+0.8% absolute) top-1 accuracy. *All methods here, and throughout the paper, use the same hardware and software implementation.*

per iteration. The mini-batch shape $B{\times}T{\times}H{\times}W$[2] (mini-batch size × number of frames × height × width) is typically *constant* throughout training. A variety of considerations go into selecting this input shape, but a common heuristic is to make the $T{\times}H{\times}W$ dimensions large in order to improve accuracy, *e.g.*, as observed in [9, 45, 47].

This heuristic is only one possible choice, however, and in general there are trade-offs. For example, one may use a smaller number of frames and/or spatial size while simultaneously increasing the mini-batch size B. With such an exchange, it is possible to process the same number of epochs (passes over the dataset) with lower wall-clock time because each iteration processes more examples. The resulting trade-off is faster training with lower accuracy.

The central idea of this paper is to avoid this trade-off— *i.e.*, to have faster training *without* losing accuracy—by making the mini-batch shape *variable* during training. By viewing the input video clips in a mini-batch as raw video signals that are sampled on a sampling grid (to be defined), we can draw a connection to *multigrid methods* for numeri-

---

[1]github.com/facebookresearch/SlowFast/blob/master/projects/multigrid

[2]We omit the channel dimension (3 for RGB) for clarity.

cal analysis [1]. These methods exploit coarse-to-fine grids to accelerate optimization. Intuitively, if we use large mini-batches with relatively small time and space dimensions (a 'coarse grid') early in training and small mini-batches with large time and space dimensions (a 'fine grid') later, then SGD may be able to scan through the data more quickly on average while finally solving for a high accuracy model, akin to how coarse grids enable solving problems on finer grids more rapidly in multigrid numerical solvers [1].

Multigrid training is possible because video models are compatible with input data of variable space and time dimensions due to weight sharing operations (*e.g.*, convolutions). In addition, CNNs are effective at learning patterns at multiple scales, *e.g.*, as observed when training with data augmentation [18, 26, 38]. We observe similar multi-scale robustness and generalization with multigrid training.

Our proposed multigrid training method is simple and effective. It is easy to implement and typically only requires small changes to a data loader. Empirically, it works with default learning rate schedules and hyper-parameters already in use. No tuning is required. Moreover, multigrid training works robustly out-of-the-box for different models (I3D [3], non-local [47], SlowFast [9]), datasets (Kinetics-400 [23], Something-Something V2 [14], and Charades [36]), initializations (random and pre-trained), and hardware scales (*e.g.*, 128 GPUs or 1 GPU). We observe a consistent speedup and performance gain in all cases without tuning. As an example, we train a SlowFast network ~4.5× faster in wall-clock time on the large-scale Kinetics dataset (Fig. 1) while also reaching a higher accuracy (+0.8% absolute). We hope these benefits provided by multigrid training will make research on video understanding more *accessible, scalable, and economical*.

2. Related Work

3D CNN video models extend 2D CNNs to model both spatial and temporal patterns. They are currently the state of the art for video understanding [3, 9, 12, 19, 30–32, 34, 43–45, 47–49]. These methods are computationally expensive, both for training and inference [43, 49]. Some recent studies propose lighter weight models that use efficient temporal modules [2, 5, 19, 21, 28–30, 32, 34, 40, 44, 46] and/or exploit temporal redundancy [9, 51]. In this paper, we show that the training time of state-of-the-art efficient models [9] can still be reduced significantly.

Efficient training can also be advanced through, *e.g.*, optimization methods (*e.g.*, [8, 24, 33, 39]), pre-training [3, 11], distributed training [13, 50], or advances in hardware [22] and software design [4, 6]. In this paper, we propose a complementary direction that exploits variable mini-batch shapes for fast training. Related to our method, Wang *et al*. [47] and Feichtenhofer *et al*. [10] initialize larger models

with smaller, fully-trained ones. These methods can potentially speed up training as well, and (as can be seen later) are a special case of multigrid training.

Multi-scale training in segmentation [16] and classification [18, 38] uses multiple image crop sizes. However, the mini-batch shape remains fixed [16,18,38]. Multigrid training on the other hand uses variable mini-batch shapes. He *et al*. [17] change the input shapes, but fix the mini-batch size. These methods shows that training with variable scales can be beneficial. Multigrid training enjoys the same property.

Multigrid methods were originally proposed for numerical boundary value problems, and later developed into an entire field in computational mathematics [1]. They typically involve iterating through cycles of coarse and fine problems, and exploit the fact that a coarse problem can be solved efficiently to speed up the overall problem solving. He and Xu [15] connect multigrid methods to deep networks through identifying the correspondence between steps in traditional multigrid methods and operators in a convolutional neural network. In this paper, we take inspiration from multigrid concepts from a more abstract view to accelerate video model training.

3. Multigrid Training for Video Models

To develop our multigrid training method we will consider a reference video model (*e.g.*, C3D [43], I3D [3]) that is trained by a baseline mini-batch optimizer (*e.g.*, SGD) that operates on mini-batches of shape $B \times T \times H \times W$ (mini-batch size × number of frames × height × width) for some number of epochs (*e.g.*, 100). The spatial-temporal shape, $T \times H \times W$, arises from resampling source videos in the training dataset according to a *sampling grid* that is specified by a temporal span, a spatial span, a temporal stride, and a spatial stride (defined in §3.1). These concepts intuitively correspond to a grid's duration/area (span) and sampling rate (stride). The baseline optimizer holds the mini-batch shape *constant* across all training iterations.

Proposed Multigrid Method. Inspired by multigrid methods in numerical analysis, which solve optimization problems on alternating coarse and fine grids, the core observation in this paper is that the underlying sampling grid that is used to train video models need not be constant during training. In fact, we will show in experiments that by *varying* the sampling grid *and* the mini-batch size during training it is possible to reduce training complexity substantially (in terms of total FLOPs and wall-clock time) while achieving similar accuracy in comparison with the baseline.

The fundamental concept that enables multigrid training is the balance between computation allocated to processing more examples per mini-batch *vs.* the computation allocated to processing larger time and space dimensions. To control

this balance, we will consider temporal and spatial shapes $t \times w \times h$ that are formed by resampling source videos with a new sampling grid that has its own spans and strides. When changing the input shape we use a scaled mini-batch size b satisfying the relation $b \cdot t \cdot h \cdot w = B \cdot T \cdot H \cdot W$, or

$$b = B\frac{T}{t}\frac{H}{h}\frac{W}{w}, \qquad (1)$$

which yields computation (in FLOPs) that is roughly equal to the computation of the aforementioned baseline mini-batch for typical 3D CNNs.[3]

Our multigrid method uses a *set of sampling grids* and a *grid schedule* that determines which grid to use in each training iteration. If training is run for a similar number of epochs regardless of the choice of grids,[4] then by making $b > B$ on average the entire training process can use fewer total FLOPs and have a lower wall-clock time.

We will experimentally investigate two questions: (i) is there a set of grids with a grid schedule that can lead to faster training without a loss in accuracy? and, (ii) if so, does it robustly generalize to new models and datasets without modification? In the following we will develop the core multigrid training concepts in detail (§3.1), provide an implementation (*i.e.*, a set of grids and a grid schedule) that work well in practice (§3.2), and then explore ablation and generalization experiments (§4).

3.1. Multigrid Training Concepts

Sampling Grids. Each video in a dataset is a discrete signal that was sampled from an underlying continuous signal generated by the physical world. The video has some number of frames and pixels per frame, which are related to the physical world by the temporal and spatial resolution of the recording device (which depends on a number of camera properties). When using one of these source videos in a training mini-batch, a sampling grid is used to *resample* it.

A sampling grid in one dimension (space or time) is defined by two quantities: a *span* and a *stride*. Their units are defined w.r.t. the source video being resampled.[5] For the time dimension, the units are frames while for the spatial dimensions the units are pixels. The span is the support size of the grid and defines the duration or area that the grid covers. The stride is the spacing between sampling points. Dividing

the span by the stride gives the number of points in the grid, which determines the shape of the input data. Note that different grids can yield the same data shape, which implies that the mini-batch size will only change (Equation (1)) if a change in the sampling grid also changes the data shape.

We note that spatial sampling grids already appear in the baseline optimizer if it uses multi-scale *spatial* data augmentation [7, 26, 37]. Under our multigrid perspective, multi-scale spatial data augmentation changes the spatial spans and strides of the resampling grid *proportionally* so that the resulting mini-batch always has the same $H \times W$ spatial shape. In contrast, we will change spans and strides by different factors, which results in a different spatial shape $h \times w$ for each grid (and likewise for the time dimension).

Grid Scheduling. We use mini-batch optimizers, which have as their most basic scheduling unit a single mini-batch iteration in which one model update is performed. The training schedule consists of some number of mini-batch iterations and is often expressed in terms of epochs. For example, training may consist of 100 or 200 epochs worth of iterations. Within this overall training schedule it is common to let the learning rate vary, such as annealing it according to a schedule defined in terms of iterations or epochs.

Scheduling other training properties is also possible. Central to our multigrid method is the idea of scheduling the sampling grids that are used throughout training. When changing grids, the mini-batch size is always scaled according to Equation (1) so that mini-batch FLOPs are held roughly constant. Grid scheduling is highly flexible, admitting a large design space from simply cycling through a sequence of pre-defined grids to using randomized grids. In §3.2 we will present a randomized, hierarchical schedule that works well in practice.

Multigrid Properties. Multigrid training relies on two properties of the data and model. First, resampling the data on different grids requires a suitable operator. For video, this operator can be a reconstruction filter applied to the source discrete signal followed by computing the values at the points specified by the grid (*e.g.*, bilinear interpolation).

Second, the model must be compatible with inputs that are resampled on different grids, and therefore might have different shapes during training. Models that are composed of functions that use weight sharing across the dimensions that are resampled, *e.g.*, 2D and 3D convolutions, recurrent functions, and self-attention, are compatible and cover most of the commonly used architectures; fully-connected layers, unless their inputs are pooled to a fixed size, are not compatible.[6] We will focus on models that use 2D and 3D con-

---

[3] In practice, the computation is not exactly equal, because of rounding (*e.g.*, w can be $\frac{W}{\sqrt{2}}$), padding, and, *e.g.*, fully connected or non-local layers. We ignore these subtleties and only use approximate FLOPs as a rough design principle. All speedups are measured by wall-clock time.

[4] In practice, a similar number of epochs (*e.g.*, within a factor of 2) are typically used for a given dataset, *even for very different models*.

[5] Between two videos these units may have different physical meanings if the videos were captured by cameras with different properties (*e.g.*, a 24 frame span from a 24 FPS video *vs.* a 24 frame span from a 30 FPS video). These properties may be unknown and therefore we define grid units with respect to source videos, not the physical world.

[6] If an appropriate operator exists for 'resampling' model *parameters* so that they are compatible with new input shapes, then these parameters may still be usable with multigrid training. This concept can be combined with weight sharing, *e.g.*, by dilating or resizing model filters to mirror the data sampling grid, though preliminary experiments did not improve results.
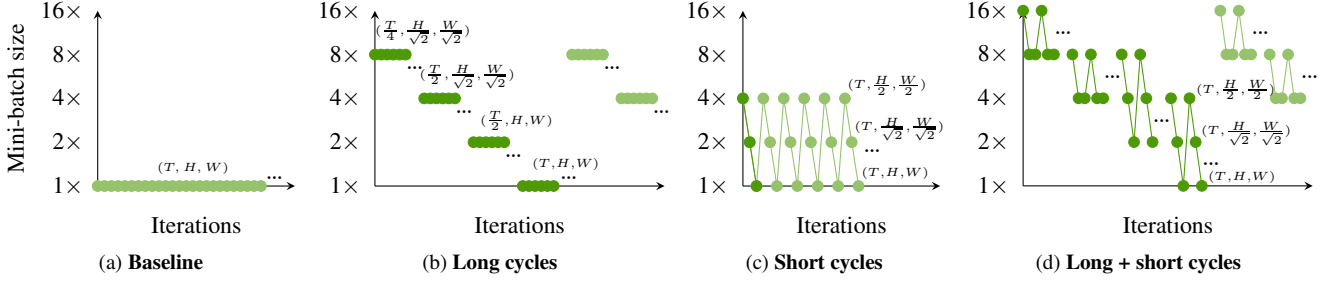
Figure 2. **A general and robust grid schedule** (§**3.2**). We contrast multigrid training with standard baseline training. (a) **Baseline** training methods typically use a fixed mini-batch shape throughout training. (b) **Multigrid long cycles** loop over inputs from small shapes (with large mini-batch sizes) to large shapes (with small mini-batch sizes), staying on each shape for several epochs. (c) **Multigrid short cycles** rapidly move through a variety of spatial shapes, changing at each iteration. (d) **Multigrid long + short cycles (our default setting)** combines long and short cycles, and moves through shapes at two frequencies simultaneously. **Dark green** points in (b), (c), and (d) correspond to one full period of a long cycle, a full short cycle, and a long+short cycle, respectively.

volutions, as well as self-attention operations in the form of non-local blocks [47]; all models end with global average pooling and a single full-connected layer as the classifier, as is common practice.

Training and Testing Distributions. The focus of this work is on multigrid methods for *training* and therefore we use a standard inference method that uses a single shape for the testing data. This choice, however, may introduce a mismatch between the data distribution used to train the model and the data distribution used at test time. To close this gap, training may be finished with some number of 'fine-tuning' iterations that use grids more closely aligned with the testing distribution, *e.g.*, see [42]. We find that this fine-tuning gives a small, but consistent improvement.

3.2. Implementation Details

Multigrid training involves a choice of sampling grids and a grid schedule, which leads to a rich design space. We use a hierarchical schedule that involves alternating between mini-batch shapes at two different frequencies: a *long cycle* that moves through a set of *base shapes*, generated by a variety of grids, staying on each shape for several epochs, and a *short cycle* that moves through a set of shapes that are 'nearby' the current base shape, staying on each one for a single iteration. This hierarchical grid schedule is described in more detail shortly and illustrated in Fig. 2.

The remainder of this subsection provides details for this design, which we have found to work well in practice. After presenting these details, we will explore what design decisions are important in ablation experiments.

Optimizer. We use SGD with momentum and a step-wise learning rate decay schedule since these are common choices in practice [9, 18, 26, 43]. Using other learning rate schedules and optimizers is also possible. Specific schedules are given in each experimental section.

Long Cycle. We use sampling grids that result in an ordered sequence of $S{=}4$ base mini-batch shapes of non-decreasing size along each dimension: $8B{\times}\frac{T}{4}{\times}\frac{H}{\sqrt{2}}{\times}\frac{W}{\sqrt{2}}$, $4B{\times}\frac{T}{2}{\times}\frac{H}{\sqrt{2}}{\times}\frac{W}{\sqrt{2}}$, $2B{\times}\frac{T}{2}{\times}H{\times}W$, and $B{\times}T{\times}H{\times}W$. These four shapes cover an intuitive range and work well in practice. The long cycle is synchronized with the stepwise learning rate decay schedule: a full cycle over the S shapes occurs exactly once for each learning rate stage. We train on each shape for the same number of iterations.

We use a simple randomized strategy to generate a mini-batch with the target input shape for each training iteration. For each video to be used in the mini-batch, we select a random span from a specified range and set the stride such that the desired shape is produced when sampling on the resulting grid. For the spatial dimensions, this strategy amounts to resizing a random crop to the desired shape using bilinear interpolation (similar to random cropping used in image classification [18,26,38]). For the temporal dimension, this strategy amounts to selecting a random temporal crop and subsampling its frames. The sampling range for spans is specified in each experimental section.

Short Cycle. The short cycle rapidly moves through a variety of spatial shapes, changing at each iteration. By default, we use the following 3-shape short cycle. For iteration i, let $m = i \pmod 3$; if $m{=}0$, then we set the spatial shape to $\frac{H}{2}{\times}\frac{W}{2}$; if $m{=}1$, we use $\frac{H}{\sqrt{2}}{\times}\frac{W}{\sqrt{2}}$; otherwise, the current base spatial shape from the long cycle is used.

The short cycle can be applied on its own or in conjunction with the long cycle. The mini-batch size is again scaled using Equation (1). The same randomized grid strategy is applied to sample data for the target mini-batch shape.

Learning Rate Scaling. When the mini-batch size changes due to the long cycle, we apply the linear scaling rule [13] to adjust the learning rate by the mini-batch size scaling factor (thus either $8{\times}$, $4{\times}$, $2{\times}$, or $1{\times}$). We found that this adjust-

ment is harmful if applied to mini-batch size changes due to the short cycle and therefore we only adjust the learning rate when the long cycle base shape changes.

Fine-tuning Phase. If the baseline optimizer uses L learning rate (LR) stages, then we apply the long and short cycles in the first $L-1$ LR stages. We use the corresponding L-th stage for fine-tuning to help match the training and testing distributions, similar to [42]. In the first half of the fine-tuning iterations we use the $L-1$-st learning rate and in the second half we use the final (L-th) learning rate. While fine-tuning we use the short cycle (as data augmentation), but not the long cycle.

Batch Normalization. The behavior of Batch Normalization (BN) [20] depends on mini-batch statistics. In traditional trainers, the constant mini-batch size is also a hyperparameter that impacts BN behaviors (*e.g.*, the noisiness of the statistics). As our multigrid method uses variable mini-batch sizes, it is desirable to *decouple* its impact on BN from that of training speedup. The following heuristic works well in practice: we compute BN statistics with a standardized *sub-mini-batch* of size 8; when the short cycle increases the overall mini-batch size by $2\times$ or $4\times$, we likewise increase the BN sub-mini-batch size to 16 and 32, respectively.

4. Experiments on Kinetics

We conduct ablation studies on the Kinetics-400 dataset [23], which is used in prior research and requires classifying each video into one of 400 categories. It contains ~240k training videos and ~20k validation videos on which we report results. Performance is measured by top-1 and top-5 accuracy.

Baseline Model and Training. We use a ResNet-50 (R50) SlowFast network [9, 18] with a 32-frame fast pathway, speed ratio $\alpha=4$, and channel ratio $\beta=1/8$ as our default model. Input frames are sampled at a temporal stride of 2.

Our baseline training recipe follows Feichtenhofer *et al.* [9]. We run synchronous SGD for 112k iterations on 128 GPUs with a mini-batch size of 4 clips per GPU (~239 epochs) with initial learning rate of 0.8. (We perform single GPU experiments in §5.) The learning rate is decreased by $10\times$ at iterations 44k, 72k, and 92k.[7] We use a weight decay of 10^{-4}, momentum of 0.9, and a linear learning rate warm-up [13] from 0.002 over 16k iterations. Input clips are random 224×224 spatial crops from clips that are randomly resized such that the shorter side $\in [256, 340]$ pixels.

At test time, we sample 10 clips per video with uniform temporal spacing and combine the predictions with average pooling following [9, 25, 47]. We use 224×224 center crop

---

[7]We use the stepwise learning rate schedule rather than the cosine schedule used in Feichtenhofer *et al.* [9] because it is still more common. Results with a cosine schedule are available in the Appendix.

testing by default [25, 48] and present results with other settings in the Appendix.

We select these training and inference procedures based on validation accuracy *using the baseline training method*. We adopt the exact same recipe for multigrid training experiments, aside from multigrid specific changes. This choice may put multigrid training at a disadvantage, but it reflects the realistic scenario in which one wants to apply multigrid training to accelerate an already known training schedule without further tuning.

Evaluation. Speedup factors are wall-clock GPU training time on P100 GPUs with CUDA 9.2 and cuDNN 7.6.3. For fair comparison, the same hardware and software implementation is used for all methods. We note that multigrid training exploits larger mini-batches, which increases data loading throughput requirements. Training may become IO bound if the data loader is not optimized appropriately or if remote data access is used. With sufficient local disk and an optimized data loader, training is typically not IO bound.

Multigrid Training Details. To sample data with spatial shape $h\times w$ that is smaller than $H\times W$, we change the default random short-side interval to $[256\frac{h}{H}, 340]$, noting that $w=h$ in our experiments. For the temporal dimension, we take t ($t<T$) frames with random stride in $[2, 2\frac{T}{t}]$.

4.1. Main Results

We compare multigrid training to baseline training in Fig. 3. In addition to the default baseline, one could speed up training by using a smaller spatial-temporal shape with a larger mini-batch size and learning rate, so we also compare to this baseline variant. For each method, we experiment with training schedules that range from $0.25\times$ to $3\times$ the number of baseline epochs (~239) to study the trade-off between training time and accuracy. Overall, multigrid training always achieves a better trade-off than baseline training. For example, multigrid training with both the long and short cycles can iterate through $1.5\times$ more epochs than baseline method, while only requiring $1/3.4\times$ the number of iterations, $1/4.5\times$ training time, and achieving higher accuracy ($75.6\% \to 76.4\%$). The wall-clock speedup is greater than the iteration reduction factor, as a larger mini-batch with smaller space/time dimensions is more parallelism-friendly on modern GPUs. Both the long and short cycles improve the trade-off and using both together performs the best.

In Fig. 3 we also observe that baseline training suffers a decline in accuracy when training for $\geq 1.5\times$ epochs. With either long and/or short cycles, a decrease in accuracy is not observed for schedules up to $2.0\times$ epochs, indicating that variable grids can help prevent overfitting.

In the following we use multigrid training with long and short cycles and $1.5\times$ more epochs than the baseline as our default since it obtains a good trade-off.
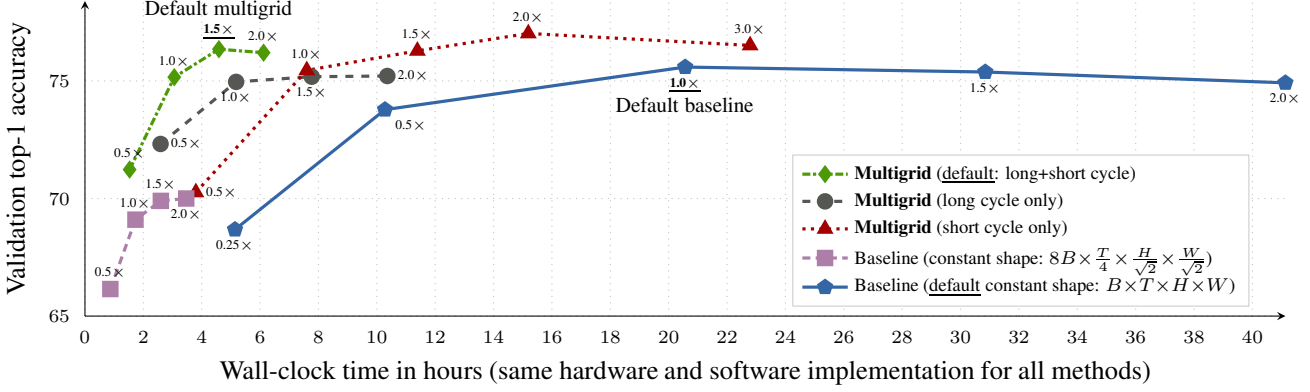
Figure 3. **Multigrid *vs*. baseline training.** Each point corresponds to one model trained with a specific schedule choice. Annotations denote training epochs relative to the baseline 1.0× schedule. For example, '1.5×' denotes training for 1.5× more epochs than the default '1.0×' baseline schedule (112k iterations or ~239 epochs). We see that **all variants of multigrid training achieve a better trade-off than baseline training, which uses a constant mini-batch shape**. Also note that multigrid training can iterate through the same number of epochs more efficiently.

4.2. Ablation Experiments

Long Cycle Design. By default, we use $S=4$ long cycle shapes with a 1.5× epoch schedule. In Table 1a, we explore using fewer shapes, where we take the last $S'<S$ shapes, for $S'\in\{1,2,3\}$. The short cycle is used in these experiments, and the $S'=1$ setting is equivalent to using the short cycle only. We run all variants for the same number of training iterations (to roughly preserve total training FLOPs), noting that methods which use fewer shapes will process fewer epochs due to having smaller mini-batches on average compared to the $S=4$ design.

We see that using each additional shape improves accuracy and saturates at $S=4$ (default). The improvement in accuracy is possibly due to the more examples seen by the model given the same amount of iterations. Compared with $S=1$ (*i.e.*, short cycle only), our default choice improves the top-1 accuracy by absolute 2.4% (74.0% → 76.4%), while being slightly faster (4.0× → 4.5×). All results use the fine-tuning phase, which we find is beneficial to varying degrees in different settings. With the default schedule, it leads to 0.4% absolute gain (76.0% → 76.4%; not shown in table).

Short Cycle Design. Adding each input shape to the short cycle leads to a clear accuracy improvement, Table 1b. Our default short cycle design (3-shape) improves over 1-shape (*i.e.*, no short cycle / long-cycle only) by absolute 1.9% (74.5% → 76.4%) in top-1 accuracy.

4.3. Generalization to Different Training Settings

Next we study how multigrid training generalizes to different training settings that are common in practice.

Pre-training. In our main results, we train models from random initialization. We see in Table 2a that with ImageNet [35] pre-training, our multigrid method obtains a

similar speedup and performance gain. (We will present more results on ImageNet-pre-trained models in §4.4.)

Temporal Shape. Next we show generalization of multigrid training for models of different temporal shapes T. We compare models that use 16-frame, 32-frame (default), and 64-frame input clips.[8] In all cases (Table 2b), multigrid training achieves a consistent accuracy gain and speedup. The 64-frame model enjoys the largest performance gain (75.9% → 77.6%) and the best speedup (5.5×).

Spatial Shape. We also demonstrate generalization of our method for models of different spatial shapes $H\times W$. We increase the baseline shape from 224×224 (default) to 320×320 and study the impact. Inference for the 320×320 model is analogous to the 224×224 case; we resize shorter side to 352 pixels and test on center 320×320 crops. In Table 2c, we see that multigrid training leads to an even larger performance gain (75.1% → 76.8%) and a more significant speedup (6.5×) in the 320×320 case. Also note with the baseline method 320×320 does not work better than 224×224, possibly due to overfitting, similar to what is reported in Tan *et al*. [41]. On the other hand, with multigrid training, spatial scaling brings improvement, possibly due to the data augmentation brought by multigrid training.

4.4. Generalization to Different Models

So far we have focused on state-of-the-art SlowFast network [9] for analysis. We next demonstrate generalization of multigrid training to different networks by presenting results using a standard R50-I3D model [3, 18] and its extension with non-local blocks (I3D-NL) [47].

---

[8]$\alpha=2$ in the 16-frame model to avoid a degenerated slow pathway.

	long cycle design	speedup	top-1	top-5
Baseline	-	-	75.6	91.9
Multigrid	1-shape (short cycle only)	4.0×	74.0	91.4
	2-shape	4.3×	75.5	92.1
	3-shape	4.4×	76.2	92.4
	4-shape (default)	**4.5×**	**76.4**	**92.4**

	short cycle design	speedup	top-1	top-5
Baseline	-	-	75.6	91.9
Multigrid	1-shape (long cycle only)	4.2×	74.5	91.6
	2-shape	4.3×	75.5	92.1
	3-shape (default)	**4.5×**	**76.4**	**92.4**

(a) **Long cycle design (with default short cycle)**   (b) **Short cycle design (with default long cycle)**

Table 1. **Ablation Study.** We perform ablations on Kinetics-400 using an R50-SlowFast network. We analyze the impact of the long cycle (Table 1a) and short cycle (Table 1b) designs. All variants of multigrid training use the same number of training iterations as our default 1.5× epoch schedule; this roughly preserves the total training FLOPs. We report wall-clock speedup relative to the baseline trained for 1.0× epochs.

	pre-train?	speedup	top-1	top-5
Baseline		-	75.6	91.9
Multigrid		**4.5×**	**76.4**	**92.4**
Baseline	✓	-	75.4	91.9
Multigrid	✓	**4.5×**	**76.0**	**92.4**

T		speedup	top-1	top-5
16	Baseline	-	74.8	91.4
16	**Multigrid**	**4.0×**	**75.2**	**91.9**
32	Baseline	-	75.6	91.9
32	**Multigrid**	**4.5×**	**76.4**	**92.4**
64	Baseline	-	75.9	92.1
64	**Multigrid**	**5.5×**	**77.6**	**93.2**

$H \times W$		speedup	top-1	top-5
224	Baseline	-	75.6	91.9
224	**Multigrid**	**4.5×**	**76.4**	**92.4**
320	Baseline	-	75.1	91.8
320	**Multigrid**	**6.5×**	**76.8**	**92.8**

(a) **Pre-training**   (b) **Temporal shape T**   (c) **Spatial shape $H \times W$**

Table 2. **Generalization Analysis.** We study how multigrid training generalizes to models both with and without ImageNet pre-training (Table 2a) and models of different temporal (Table 2b) and spatial (Table 2c) shapes. All experiments use R50-SlowFast with results on Kinetics-400. We use the default setting for multigrid training (1.5× more epochs, corresponding to 3.4× fewer iterations than baseline) in all settings. We observe that the default choice brings consistent speedup and performance gain in all cases.

Implementation Details. Both models are ImageNet-pre-trained with 3D convolutions inflated from 2D convolutions following common practice [3, 10, 47]. Each input clip consists of 16 frames, sampled at a stride of 4. I3D-NL additionally contains 5 (dot product) non-local blocks [47] in res$_3$ and res$_4$ stages. The exact model specification is given in the Appendix.

The baseline recipe trains for 100k iterations using 128 GPUs, with a mini-batch size of 2 clips per GPU (~106 epochs) and a learning rate of 0.04, which is decreased by a factor of 10 at iteration 37.5k and 75k. We do not use learning rate warm-up [13] following prior work [47]. Other training details are analogous to SlowFast training. We note again that this training recipe is selected to be the best for the baseline training method and we apply multigrid training on top without further tuning.

Evaluation. We summarize the results in Table 3. For both I3D and I3D-NL, multigrid training with the default schedule (1.5× epoch) obtains similar or better accuracy, while being up to 3.9× faster. We also experiment with a shorter baseline schedule ('baseline $\frac{1}{3.3}$' in table), which trains for the same number of iterations as the multigrid training. The shorter baseline schedule obtains a lower accuracy (3.7% and 3.2% absolute top-1 lower than multigrid). We also see that I3D-NL has a lower speedup than I3D. This is in part due to the less optimized NL operator than convolution, consuming a large portion of the training time. We ob-

model		speedup	top-1	top-5
I3D	Baseline	-	74.4	91.4
I3D	Baseline $\frac{1}{3.3}$	3.3×	71.1	89.9
I3D	**Multigrid**	**3.9×**	**74.8**	**91.7**
I3D-NL	Baseline	-	**75.5**	92.1
I3D-NL	Baseline $\frac{1}{3.3}$	3.3×	72.3	90.6
I3D-NL	**Multigrid**	**3.3×**	**75.5**	**92.4**

Table 3. **Kinetics-400 accuracy with I3D and I3D-NL.** While developed on SlowFast [9], multigrid training provides a consistent speedup and performance gain with I3D [3] and I3D-NL [47].

serve consistent improvements with larger backbone models (R101); see the Appendix.

5. Case Study: 1-GPU Training on Kinetics

Our experiments thus far use a large number of GPUs (128) in parallel. However, a more common training recipe may use far fewer GPUs (*e.g.*, 1 to 8) and given that one of our goals is to make video research more accessible by reducing computational requirements it is important to explore the application of our multigrid method in the few-GPU regime, without any tuning.

As a case study, we use a *single GPU* to train an I3D model on Kinetics-400 using the quick training recipe from the public repository[9] of Wang *et al.* [47]. We apply multigrid training on top without further tuning. This schedule

---

[9] https://github.com/facebookresearch/video-nonlocal-net

	training time (days)	top-1	top-5
Baseline	6.7	72.5	90.4
Multigrid	**2.0**	72.5	90.4

Table 4. **Case study: 1-GPU training on Kinetics-400.** Multigrid training reduces the training time from nearly 1 week to 2 days on a single GPU. We hope the reduced training time will make video understanding research more accessible and economical.

trains for 1200k iterations (after adjusting with the linear scaling rule [13]) on one GPU with 8 clips (~40 epochs). The learning rate is 0.00125, which is decreased by a factor of 10 at iteration 600k and 1000k. Dropout and random scaling are disabled to accelerate convergence given the short schedule. Each input clip consists of 8 frames, sampled at a stride of 8, when using the baseline optimizer. Other training details are the same as the I3D experiments.

Table 4 shows that multigrid training generalizes well out-of-the-box to a few-GPU, short-schedule setting. With multigrid training we are able to achieve 72.5% (73.1% with 30-crop testing [47]) top-1 accuracy *in 2 days using only 1 GPU*, while the baseline method would need nearly 1 week. (When using a small model, we observe a smaller wall-clock speedup of ~3.3× compared to a larger model, which typically yields a ~4.5× speedup). We hope the reduced training time with multigrid training will make video understanding research more accessible and economical.

6. Experiments on Something-Something V2

We next evaluate multigrid training on the Something-Something V2 dataset [14], which contains 169k training, and 25k validation videos. Each video shows an interaction with everyday objects. The task is classification with 174 action classes. Performance is evaluated by top-1 and top-5 accuracy. This task is known to require more 'temporal modeling' to solve than Kinetics [49].

Implementation Details. We use an R50-SlowFast model [9,18] with 64-frame fast pathway with speed ratio $\alpha=4$ and channel ratio $\beta=1/8$. The model is pre-trained on Kinetics-400 following prior work [29]. The baseline training recipe trains for 230k iterations on 8 GPUs, with a mini-batch size of 2 clips per GPU and a learning rate of 0.03, which is decreased by a factor of 10 at iteration 150k and 190k. Other training details are analogous to Kinetics experiments; see the Appendix for details.

Results. Similar to what we observe on Kinetics, multigrid training obtains a better trade-off than baseline training on Something-Something V2 (Table 5). With the default 1.5×-epoch training, multigrid training is 5.6× faster while obtaining a slightly higher accuracy. Multigrid training behaves consistently for the 'spatial heavy' Kinetics dataset and the 'temporal heavy' Something-Something V2 dataset.

	speedup	top-1	top-5
Baseline	-	$60.9_{\pm 0.31}$	$87.2_{\pm 0.13}$
Baseline $\frac{1}{5.2}$	5.2×	$54.6_{\pm 0.13}$	$83.0_{\pm 0.14}$
Multigrid 1.0×	**8.3×**	$60.0_{\pm 0.31}$	$86.8_{\pm 0.05}$
Baseline $\frac{1}{3.4}$	3.4×	$57.3_{\pm 0.13}$	$84.7_{\pm 0.15}$
Multigrid 1.5× (default)	5.6×	$61.2_{\pm 0.18}$	$87.4_{\pm 0.12}$
Baseline $\frac{1}{2.6}$	2.6×	$58.7_{\pm 0.06}$	$85.8_{\pm 0.15}$
Multigrid 2.0×	4.2×	$\mathbf{61.7}_{\pm 0.20}$	$\mathbf{87.8}_{\pm 0.12}$

Table 5. **Results on Something-Something V2.** Multigrid training achieves a better trade-off than baseline training. Results are the mean and standard deviation over 5 runs.

	speedup	mAP (%)
Baseline	-	$38.0_{\pm 0.18}$
Baseline $\frac{1}{5.3}$	5.3×	$27.5_{\pm 0.15}$
Multigrid 1.0×	**8.6×**	$36.8_{\pm 0.31}$
Baseline $\frac{1}{3.5}$	3.5×	$31.5_{\pm 0.26}$
Multigrid 1.5× (default)	5.7×	$\mathbf{38.2}_{\pm 0.06}$
Baseline $\frac{1}{2.6}$	2.6×	$33.6_{\pm 0.13}$
Multigrid 2.0×	4.3×	$37.4_{\pm 0.15}$

Table 6. **Results on Charades.** Multigrid training shows consistent speedups compared with the other datasets. Results are the mean and standard deviation over 5 runs.

7. Experiments on Charades

We finally evaluate our method on the Charades dataset [36], which is relatively small, consisting of only 9,848 videos in 157 action classes. The task is to predict all actions in a video. Performance is measured by mAP.

Implementation Details. We use the same R50-SlowFast model [9, 18], with the same Kinetics pre-training as the Something-Something experiments. Training details are available in the Appendix.

Results. Overall we observe consistent results compared with Kinetics and Something-Something V2 (Table 6). The default multigrid training is 5.7× faster, while achieving slightly better mAP. Overall, we see that even for the smaller Charades dataset, with strong large-scale pretraining, multigrid training is beneficial.

8. Conclusion

We propose a multigrid method for fast training of video models. Our method varies the sampling grid and the mini-batch size during training, and can process the same number of epochs using a small fraction of the computation of the baseline trainer. With a single *out-of-the-box* setting, it works on multiple datasets and models, and consistently brings a ~3-6× speedup with comparable or higher accuracy. It works across a spectrum of hardware settings from 128 GPU distributed training to single GPU training. We hope the reduced training time will make video understanding research more accessible, scalable, and economical.

A. Appendix

A.1. Supplementary Experiments

R101-SlowFast Results. We demonstrate generalization of multigrid training to deeper backbones by extending our default R50-SlowFast network to R101-SlowFast. All other designs and training procedures remain unchanged.

backbone		speedup	top-1	top-5
R50 (default)	Baseline	-	75.6	91.9
R50 (default)	**Multigrid**	**4.5×**	**76.4**	**92.4**
R101	Baseline	-	76.5	92.4
R101	**Multigrid**	**4.4×**	**77.0**	**92.9**

As expected, R101-SlowFast outperforms R50-SlowFast and we observe a consistent speedup and accuracy gain over the baseline with multigrid training.

Long Cycle Design. By default we use multiple long cycles that are synchronized with the stepwise learning rate (LR) schedule (*i.e.*, one long cycle period per LR stage). We compare our default design ('multi-cycle') with an alternative that uses only a single long cycle period ('single-cycle') throughout all of training. Note that the single-cycle design does not use a fine-tuning phase as it is unclear how to incorporate it into this design.

	long cycle design	speedup	top-1	top-5
Baseline	-	-	75.6	91.9
Multigrid	single-cycle	5.2×	74.4	91.8
	multi-cycle (default)	**4.5×**	**76.4**	**92.4**

We observe that our default, multi-cycle design works better. In the multi-cycle design, the later shapes, which are closer to the final testing distribution, are used with each LR. We conjecture that exposing the model to these shapes with the larger (earlier) LRs is important for generalizing to the testing distribution. In contrast, the single-cycle design only uses the later shapes with relatively low LRs.

Cosine Learning Rate Schedule. We develop multigrid training assuming a stepwise LR schedule. Next we experiment with a cosine LR schedule. We experiment with both the multi-cycle and the single-cycle design for long cycles. *No further modifications* are applied to multigrid training.

LR schedule		speedup	top-1	top-5
Stepwise (default)	Baseline	-	75.6	91.9
	Multigrid	**4.5×**	**76.4**	**92.4**
Cosine	Baseline	-	75.8	92.0
	Multigrid (single long cyc.)	5.2×	75.4	92.1
	Multigrid (multi long cyc.)	4.2×	75.3	92.1

We observe that multigrid training on a cosine schedule obtains a slightly lower accuracy than the default stepwise schedule. The lower accuracy is possibly due to the relatively smaller learning rates used in larger (later) shapes as the LR is monotonically decreasing in a cosine schedule. However, it still obtains a consistent speedup and a comparable accuracy to baseline, suggesting robustness of the multigrid strategy. The two long-cycle designs obtain a similar accuracy.

Testing Settings Next we present results with additional test-time settings that are common in the literature. Here we use the 64-frame R50-SlowFast due to its high accuracy. Our multigrid method trains this model 5.5× faster than the baseline.

	center 224^2		3-crop 224^2		3-crop 256^2	
	top-1	top-5	top-1	top-5	top-1	top-5
Baseline	75.9	92.1	76.5	92.2	77.2	92.5
Multigrid	**77.6**	**93.2**	**78.1**	**93.5**	**78.1**	**93.4**

As expected, using 3-crop (left-center-right) testing improves accuracy for both baseline and multigrid training.

A.2. Supplementary Implementation Details

The I3D and I3D-NL architectures used in generalization analysis are shown below (assuming $16 \times 224 \times 224$ inputs):

Layer	Specification		Output size
$conv_1$	$1 \times 7 \times 7$, 64, stride 1, 2, 2		$16 \times 112 \times 112$
$pool_1$	$1 \times 3 \times 3$ max, stride 1, 2, 2		$16 \times 56 \times 56$
res_2	$1 \times 1 \times 1$, 64 $1 \times 3 \times 3$, 64 $1 \times 1 \times 1$, 256	$\times 3$	$16 \times 56 \times 56$
res_3	$1 \times 1 \times 1$, 128 $1 \times 3 \times 3$, 128 $1 \times 1 \times 1$, 512	$\times 4$	$16 \times 28 \times 28$
res_4	$3 \times 1 \times 1$, 256 $1 \times 3 \times 3$, 256 $1 \times 1 \times 1$, 1024	$\times 6$	$16 \times 14 \times 14$
res_5	$3 \times 1 \times 1$, 512 $1 \times 3 \times 3$, 512 $1 \times 1 \times 1$, 2048	$\times 3$	$16 \times 7 \times 7$

'I3D-NL' additionally uses non-local operators [47] after blocks 1 and 3 of res_3, and blocks 1, 3, and 5 of res_4.

Something-Something V2 Training. We use a linear warm-up [13] for 2k iterations from 0.0001 and a weight decay of 10^{-6}. As Something-Something V2 requires distinguishing between directions, we disable random flipping during training. Following [29], we use segment-based input frame sampling, *i.e.*, we split each video into segments, and from each of them, sample one frame to form a clip.

Charades Training. The baseline method trains for 28k iterations with a learning rate of 0.0375, which is decreased by a factor of 10 at iteration 20k and 24k.

References

[1] WF Briggs, VE Henson, and Stephen F McCormick. *A Multigrid Tutorial, 2nd Edition*. SIAM, 2000. 2

[2] Joao Carreira, Viorica Patraucean, Laurent Mazare, Andrew Zisserman, and Simon Osindero. Massively parallel video networks. In *ECCV*, 2018. 2

[3] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017. 2, 6, 7

[4] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. TVM: An automated end-to-end optimizing compiler for deep learning. In *OSDI*, 2018. 2

[5] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. Multi-fiber networks for video recognition. In *ECCV*, 2018. 2

[6] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 2

[7] Dan C Cireşan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011. 3

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011. 2

[9] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. SlowFast networks for video recognition. In *ICCV*, 2019. 1, 2, 4, 5, 6, 7, 8

[10] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, 2016. 2, 7

[11] Deepti Ghadiyaram, Du Tran, and Dhruv Mahajan. Large-scale weakly-supervised pre-training for video action recognition. In *CVPR*, 2019. 2

[12] Rohit Girdhar, Joao Carreira, Carl Doersch, and Andrew Zisserman. Video action transformer network. In *CVPR*, 2019. 2

[13] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 2, 4, 5, 7, 8, 9

[14] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The "Something Something" video database for learning and evaluating visual common sense. In *ICCV*, 2017. 2, 8

[15] Juncai He and Jinchao Xu. MgNet: A unified framework of multigrid and convolutional neural network. *Science China Mathematics*, 2019. 2

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 2

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *PAMI*, 2015. 2

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 4, 5, 6, 8

[19] Noureldien Hussein, Efstratios Gavves, and Arnold WM Smeulders. Timeception for complex action recognition. In *CVPR*, 2019. 2

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

[21] Boyuan Jiang, MengMeng Wang, Weihao Gan, Wei Wu, and Junjie Yan. STM: Spatiotemporal and motion encoding for action recognition. In *ICCV*, 2019. 2

[22] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017. 2

[23] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The Kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 2, 5

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[25] Bruno Korbar, Du Tran, and Lorenzo Torresani. SCSampler: Sampling salient clips from video for efficient action recognition. In *ICCV*, 2019. 5

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 2, 3, 4

[27] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989. 1

[28] Myunggi Lee, Seungeui Lee, Sungjoon Son, Gyutae Park, and Nojun Kwak. Motion feature network: Fixed motion filter for action recognition. In *ECCV*, 2018. 2

[29] Ji Lin, Chuang Gan, and Song Han. Temporal shift module for efficient video understanding. In *ICCV*, 2019. 2, 8, 9

[30] Chenxu Luo and Alan L Yuille. Grouped spatial-temporal aggregation for efficient action recognition. In *ICCV*, 2019. 2

[31] Brais Martinez, Davide Modolo, Yuanjun Xiong, and Joseph Tighe. Action recognition with spatial-temporal discriminative filter banks. In *ICCV*, 2019. 2

[32] AJ Piergiovanni, Anelia Angelova, Alexander Toshev, and Michael S Ryoo. Evolving space-time neural architectures for videos. In *ICCV*, 2019. 2

[33] Siyuan Qiao, Zhe Lin, Jianming Zhang, and Alan L Yuille. Neural rejuvenation: Improving deep network training by enhancing computational resource utilization. In *CVPR*, 2019. 2

[34] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatiotemporal representation with pseudo-3D residual networks. In *ICCV*, 2017. 2

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015. 6

[36] Gunnar A Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*, 2016. 2, 8

[37] Patrice Y Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, 2003. 3

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2, 4

[39] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. In *ICLR*, 2018. 2

[40] Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *ICCV*, 2015. 2

[41] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 6

[42] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy. In *NeurIPS*, 2019. 4, 5

[43] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3D convolutional networks. In *ICCV*, 2015. 2, 4

[44] Du Tran, Heng Wang, Lorenzo Torresani, and Matt Feiszli. Video classification with channel-separated convolutional networks. In *ICCV*, 2019. 2

[45] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, 2018. 1, 2

[46] Limin Wang, Wei Li, Wen Li, and Luc Van Gool. Appearance-and-relation networks for video classification. In *CVPR*, 2018. 2

[47] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018. 1, 2, 4, 5, 6, 7, 8, 9

[48] Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krähenbühl, and Ross Girshick. Long-term feature banks for detailed video understanding. In *CVPR*, 2019. 2, 5

[49] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *ECCV*, 2018. 2, 8

[50] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. ImageNet training in minutes. In *ICPP*, 2018. 2

[51] Mohammadreza Zolfaghari, Kamaljeet Singh, and Thomas Brox. ECO: Efficient convolutional network for online video understanding. In *ECCV*, 2018. 2