Received: 31 May 2021 | Revised: 26 August 2021 | Accepted: 12 September 2021

DOI: 10.1002/spe.3035

RESEARCH ARTICLE

WILEY

A computational approach for progressive architecture shrinkage in action recognition

Matteo Tomei[1] | **Lorenzo Baraldi**[1] | **Giuseppe Fiameni**[2] | **Simone Bronzin**[3] | **Rita Cucchiara**[1]

[1]University of Modena and Reggio Emilia, Modena, Italy

[2]NVIDIA AI Technology Center, Bologna, Italy

[3]Metaliquid Srl, Milano, Italy

Correspondence
Matteo Tomei, University of Modena and Reggio Emilia, Modena, Italy.
Email: matteo.tomei@unimore.it

Abstract

Efficiency plays a key role in video understanding modeling, and developing more efficient spatiotemporal deep networks is a key ingredient for enabling their usage in production scenarios. In this work, we propose a methodology for reducing the computational complexity of a video understanding backbone while limiting the drop in accuracy caused by architectural changes. Our approach, named, Progressive Architecture Shrinkage, applies a sequence of reduction operators to the hyperparameters of a network to reduce its computational footprint. The choice of the sequence of operations is automatically optimized in a coordinate-descent schema, and the approach transfers knowledge from both the initial network and previous stages of the shrinking process by employing a Knowledge Distillation and an adaptive fine-tuning strategy. As each iteration of the shrinking algorithm requires to train a large-scale video understanding network, we perform experiments on MARCONI 100—a supercomputer equipped with an IBM Power9 architecture and Volta NVIDIA GPUs. Experimental evaluations are conducted using two backbones and three different action recognition benchmarks. We show that, through our approach, high accuracy levels can be maintained while reducing the number of multiply–adds operations by four times with respect to the original architectures. Code will be made available.

KEYWORDS

architectural optimization, distributed training, video understanding

1 | INTRODUCTION

The last few years have witnessed a relevant interest in developing novel and more accurate video understanding models based on deep learning. Defining and training models which can effectively extract spatiotemporal features from an input video is indeed a crucial challenge toward AI systems that can encode and understand motion, actions, and interactions between people and objects.[1] Such abilities are required to process the huge amount of video data that is uploaded every day on video-sharing and social network platforms, as well as to endow novel surveillance systems with the

ability to understand and predict actions. Starting from models that adapted 2D CNNs to the spatiotemporal case,[2,3] the research community has over time developed more principled convolutional operators[4-6] and architectural choices,[7,8] gaining considerable advances in terms of effectiveness and accuracy.

Being videos sequences of consecutive frames, the development of video models requires significant computational efforts with an impact on the cost of each experiment and, ultimately, on the speed of the research pace. While the computational cost of researching novel architectures is well known and has been properly managed by the research community in the last years, the size and energy requirements of state of the art networks still severely limit their applicability in production environments. Endowing any content sharing platform with real-time video analysis with a state of the art spatiotemporal network, for example, would be almost unsustainable in the majority of cases.

In the attempt to make a step forward in the development of more sustainable spatiotemporal models, in this article we devise a generic strategy for turning any video model into a more efficient one, limiting the loss in accuracy caused by the reduction of computational demands. We see the process of shrinking a spatiotemporal network as that of applying a sequence of "reduction" operations over the factors which affect its computational complexity and sustainability (Figure 1). For instance, one could reduce the computational footprint by halving the number of channels of the initial architecture, and then reduce the spatial size of the input. In the same manner, one could gain a similar improvement in efficiency by increasing the temporal stride of the input and then halving the number of channels. While both these choices would increase the efficiency of the resulting network, surely they would have a different impact in terms of the final accuracy of the two architectures when trained on a dataset of choice. Our approach, named, Progressive Architecture Shrinkage (PAS), sequentially shrinks a base network by selecting an optimal sequence of reduction operators, so to lower the computational complexity while limiting losing accuracy.

To maintain high accuracy levels after the application of each reduction operator, we employ a Knowledge Distillation paradigm that aims at preserving the knowledge learned in the initial network. Further, we transfer knowledge between each reduction step by applying an adaptive fine-tuning strategy. The resulting approach is general enough to be applied to any video backbone and, after a sequence of reduction steps, produces a smaller network with a computational complexity of choice. Although the overall approach requires a high computational load, since at each iteration the reduced network needs to be retrained, PAS achieves impressive results in finding a good trade-off in computational complexity and accuracy.

To validate the effectiveness of our approach, we perform experiments by shrinking two implementations of recently proposed backbones, namely, R(2+1)D[6] and SlowFast,[7] when training on the Kinetics-400 dataset.[1] Further, we also assess the capabilities of the reduced networks on UFC101[9] and HMDB51.[10]

Although recently there has been a growing interest in efficient video processing, and architectural modification have been investigated in literature,[8,11] to the best of our knowledge this is the first paper to employ a sequential shrinkage approach to reduce the computational complexity of an existing network. While we chose to focus on action recognition[12] because of its elevated computational requirements, we notice that our approach is general and could in principle be applied to any deep neural network. Because of the significant reduction in computational load it can generate, the proposed approach could benefit several other application scenarios in which computational resources are limited or real-time processing is mandatory, like Edge AI[13] or medical imaging.[14]

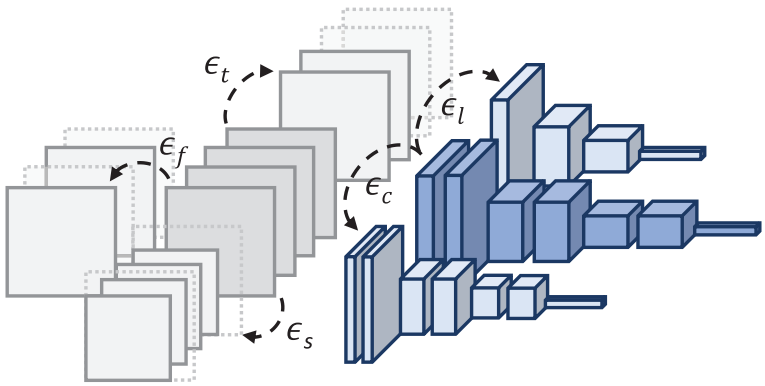

FIGURE 1 Conceptual overview of our approach. We progressively shrink a video understanding model by modifying its input and architectural hyperparameters through a set of reduction operators (in figure, outlined with ϵ_*), minimizing the loss in accuracy during the overall process—so to obtain a smaller but effective model

Contributions. To sum up, our contributions are as follows:

- We propose a Progressive Architecture Shrinkage approach for lowering the computational demands of video networks. Our method is based on a coordinate descent optimization of a sequence of reduction operations, which gradually shrink an input network.
- To reduce the loss in accuracy caused by architectural modifications, we devise a Knowledge Distillation and an adaptive fine-tuning strategy to retain knowledge from the base network and previous iterations.
- We extensively evaluate our approach using two popular video backbones on Kinetics-400, where we show that our approach can scale down the required FLOPs by a factor of four without a significant accuracy loss.
- Finally, we test the transfer capabilities of the learned networks on both UCF101 and HMDB51.

2 | RELATED WORK

Video understanding networks. Recent advances in video understanding models on the Kinetics dataset[1] are retracing the history of 2D backbones on ImageNet.[15,16] Being convolutional networks the standard approach to challenge video-related tasks, 3D CNNs are usually inspired by their 2D counterparts, with the additional burden of having to manage the temporal dimension. Recent solutions extend kernels to handle both spatial and temporal information together,[2,17,18] decompose them to spatial and temporal ones,[6,19-21] or even consider two separate but identical networks to learn appearance and motion separately.[3] As one of the most common motion descriptors is the optical flow,[22] which can provide a dense estimation of apparent motion, it is common to design a stream for RGB data and one for optical flow.[17,23,24] Recent SlowFast networks,[7] further, have demonstrated that the same RGB input can be processed by a high frame-rate network and a low frame-rate one to achieve competitive results. Other works[25-27] focus on temporal modeling, exploit attention-like[28] operators to find spatiotemporal relationships, or adopt Neural Architecture Search techniques.[29,30]

Efficient 3D video architectures. Recently, there has been a growing interest in efficient video processing.[11] Speed-accuracy trade-off was analyzed by Xie et al.,[4] where early 3D convolutions were replaced by 2D ones in the network design. A policy network to decide per-frame input resolution has been proposed by Meng et al.,[31] improving efficiency when handling less informative frames. Channel-separated convolutions for video classification have been explored by Tran et al.[32] Temporal shift module[5] shifts channels along the temporal dimension and can be inserted into 2D CNN for temporal modeling without extra computation. X3D networks[8] progressively expand a 2D CNN through a form of coordinate descent in the space defined by some expansion axes, achieving impressive computation/accuracy trade-offs. The importance of efficiency in video-related tasks is highlighted by the huge efforts in this direction.[20,33-38] Our proposed approach starts from any existing architecture, and progressively reduces its computation requirements while preserving accuracy at most.

Knowledge Distillation for video modeling. Knowledge Distillation (KD) has been first proposed by Hinton et al.,[39] and many alternatives followed.[40] The idea is to train a small network, the student, using the knowledge from a pretrained bigger model, the teacher. While many approaches have been proposed for applying KD in image-related tasks,[41-45] by transferring logits or intermediate features, the same does not hold for video-related tasks, where only a few methods have been introduced. A student looking only at a small fraction of the input frames has been proposed by Bhardwaj et al.[46] Other works have exploited KD for multimodal action recognition,[47-49] transferring knowledge between networks trained on different modalities. In this work, we propose to train progressively reduced students using KD to retain the knowledge of the initial spatiotemporal network.

3 | PROPOSED APPROACH

We propose a methodology that progressively reduces the computational needs of a given video architecture by modifying either its input size or architectural hyperparameters while minimizing the impact of these modifications on the resulting accuracy of the network. Starting from an initial network, this is achieved following an iterative approach, requiring large scale and parallel computing during training, but providing a smaller model with far fewer resource requirements for testing in the end.

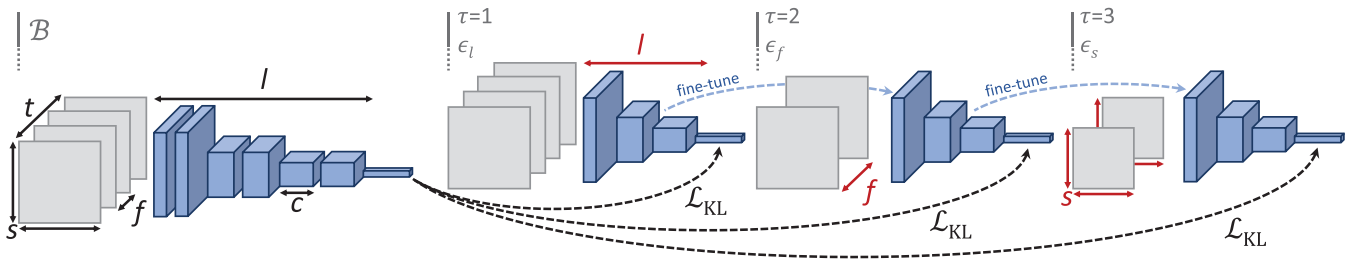

FIGURE 2 Example of a sequence of reduction operations obtained with Progressive Architecture Shrinkage (PAS). At each iteration, PAS selects a reduction operator ϵ which is applied over the current network. The latter is trained with a KD criterion with respect to the base network $\mathcal{B}$ and with an adaptive fine-tuning when possible

Given the initial network $\mathcal{B}$, at each iteration τ the procedure selects a reduction operator ϵ_τ that can alter either the input size or the architectural parameters of the current network, and which produces a reduced network $\mathcal{R}_\tau = \epsilon_\tau(\mathcal{R}_{\tau-1})$. The choice of the sequence of reduction operations is optimized by following a coordinate descent schema which aims at maximizing a trade-off between computational demands and accuracy. While the new reduced network is trained on the same dataset on which $\mathcal{B}$ was trained, to limit the loss of accuracy we both distill the activations of the base model and apply a sequential fine-tuning strategy. This is visually depicted in Figure 2, where we provide an overview of the approach and of the sequence of reduction operations. The sequential shrinkage of the network continues until a satisfactory computational complexity is reached.

3.1 | Reduction operations

The structure of a spatiotemporal convolutional network is defined by both the shape of its input, in terms of the number of frames and their spatial resolution, and architectural hyperparameters, for example, the number of layers in the network and their filters. In the following, we define a basic set of reduction operations that are used to sequentially modify the network's structure at each iteration. All these operations have an impact on the input or on the architecture. Since these belong to integer arithmetic, we omit rounding stages for clarity in the following.

- ϵ_s reduces the spatial resolution of the input by spatially downsampling the video clip by a factor γ_s. Given an input clip shape of $(\mathsf{T}, \mathsf{C}, \mathsf{H}, \mathsf{W})$, where T is the number of frames in the video clip, C the number of RGB channels for each frame, H the frame's height, and W the frame's width, this operation returns an output shape of $(\mathsf{T}, \mathsf{C}, \mathsf{H}/\gamma_s, \mathsf{W}/\gamma_s)$, where the two spatial axes have been downsampled through a resize operation.
- ϵ_t reduces the temporal length of the input, by cutting the input sequence up to a length proportional to γ_t. Given the same shape of the original clip, this operation returns an output shape of $(\mathsf{T}/\gamma_t, \mathsf{C}, \mathsf{H}, \mathsf{W})$, where the new tensor only contains the first T/γ_t frames of the original tensor.
- ϵ_f reduces the frame rate of the clip by increasing its temporal stride. Given an input clip shape of $(\mathsf{T}, \mathsf{C}, \mathsf{H}, \mathsf{W})$, this operations again returns an output shape of $\left(\mathsf{T}/\gamma_f, \mathsf{C}, \mathsf{H}, \mathsf{W}\right)$, where the new tensor is obtained from the first one by sampling a frame every γ_f.
- ϵ_c reduces the number of output channels of all the convolutional layers of the network by a factor of γ_c. Given a convolutional layer with C filters, the application of this operation amounts to setting the number of filters of the same layer to C/γ_c.
- ϵ_l reduces the number of layers in the network by a factor of γ_l. As modern ResNet-like networks are organized as sequences of convolutional blocks, we implement this operation as a reduction of the number of layers inside each block—so that the resulting network maintains the original architectural choices while having fewer layers.

3.2 | Progressive architecture shrinkage

The goal of progressive architecture shrinkage is to find the best sequence of reduction operations $\left(\epsilon_1^*, \ldots, \epsilon_T^*\right)$, with $\epsilon_i^* \in \{\epsilon_s, \epsilon_t, \epsilon_f, \epsilon_c, \epsilon_l\}$, to be applied to the base network $\mathcal{B}$ in order to reduce its computational cost and keep its effectiveness

unaltered as much as possible. To jointly take into account these objectives, we define the quality of a network as the ratio between its accuracy and computational cost.

Following previous works on video recognition,[6-8,28,32] we measure the efficiency of a network as the number of floating point operations it requires to process a single sample during the evaluation phase, and evaluate the effectiveness of the network on a given benchmark through its top-1 classification accuracy on the validation set.

Taking inspiration from previous works[8,50] we optimize the shrinkage objective by applying a form of coordinate descent in the hyperparameter space defined by the reduction axes. At each iteration, given the current network $\mathcal{R}_\tau$ (where $\mathcal{R}_0$ corresponds to the base network $\mathcal{B}$) we explore a number of hypotheses equal to the number of reduction axes, each of them obtained by applying a reduction operator to $\mathcal{R}_\tau$. We then select the hypothesis that maximizes the ratio between the reduction in computational cost and the reduction in accuracy. This amounts to selecting the reduction operator ϵ_τ^* that satisfies the following:

$$\epsilon_\tau^* = \arg\max_{\epsilon_j} \frac{C(\mathcal{B}) - C(\epsilon_j(\mathcal{R}_{\tau-1}))}{\text{Acc}(\mathcal{B}) - \text{Acc}(\epsilon_j(\mathcal{R}_{\tau-1}))}, \quad \epsilon_j \in \{\epsilon_s, \epsilon_t, \epsilon_f, \epsilon_c, \epsilon_l\}, \tag{1}$$

where $C(\cdot)$ indicates the number of floating points operations required by a network to process a single sample, $\text{Acc}(\cdot)$ indicates the top-1 validation accuracy of a network, and $\mathcal{R}_{\tau-1}$ is the reduced network obtained at the previous iteration, defined as

$$\mathcal{R}_{\tau-1} = \epsilon_{\tau-1}^*(\epsilon_{\tau-2}^*(...\epsilon_2^*(\epsilon_1^*(\mathcal{B})))) \tag{2}$$

being ϵ_i^* the reduction operation chosen at iteration i, and $\epsilon_j(\cdot)$ the application of a reduction operator over a network.

Exploring each hypothesis requires to retrain a new network, and each step of the coordinate descent procedure described above requires to independently train a number of networks equal to the number of reduction operators. As opposed to a recent work by Feichtenhofer et al.,[8] which investigated the use of coordinate descent for progressively increasing the size of a model, in our case the size and computational cost of the models decrease at each iteration.

Applying a constant computational complexity scaling factor. To ensure that the steps taken in the coordinate descent approach are consistent along each direction, we design our reduction operations so to keep a constant complexity reduction factor between the application of two subsequent reduction operations whenever possible. Under the hypothesis that $C(\epsilon_j(\mathcal{R}_{\tau-1}))/C(\mathcal{R}_{\tau-1})$ is constant, the rule for selecting the best hypothesis (see Equation 1) can be reduced to simply selecting the hypothesis with maximum accuracy, that is,

$$\epsilon_\tau^* = \arg\max_{\epsilon_j} \text{Acc}(\epsilon_j(\mathcal{R}_{\tau-1})), \ \epsilon_j \in \{\epsilon_s, \epsilon_t, \epsilon_f, \epsilon_c, \epsilon_l\}. \tag{3}$$

In practice, we apply reduction operators which lead to a complexity reduction factor $C(\epsilon_j(\mathcal{R}_{\tau-1}))/C(\mathcal{R}_{\tau-1})$ roughly equal to 2 (e.g., halving the temporal resolution of a model usually leads to halving the FLOPs required by a model). It shall be noted, however, that the reduction in computational complexity depends on both the reduction operator and on the architecture of the current network. For instance, when the temporal resolution of the reduced model becomes lower than the total temporal downsampling factor of the network, the temporal resolution of the activation maps in the network tail will become equal to 1. In this case, halving the temporal resolution would reduce the computational complexity by a factor ≤ 2. We choose to maintain the reduction operators unaltered and opt for Equation (1) for choosing the best reduction operation in this case.

3.3 | Training via distilling the knowledge

As the reader will have noticed, the base network $\mathcal{B}$ has not been involved in the optimization process until now, except for being employed as the architectural starting point for a sequence of progressively reduced networks. At each iteration τ, we want to keep the knowledge from $\mathcal{B}$ as much as possible, while cutting down the complexity of $\mathcal{R}_\tau$ with respect to the network produced at the previous iteration, $\mathcal{R}_{\tau-1}$. Therefore, when training a reduced network $\mathcal{R}_\tau$, we distill knowledge[39]

from the base network $\mathcal{B}$ to the current reduced network. Knowledge Distillation[39] has recently emerged as a powerful technique to transfer knowledge from large models to smaller ones: these two roles are fulfilled by the base network $\mathcal{B}$ and the progressively reduced networks $\mathcal{R}_\tau$, respectively.

The process of transferring knowledge from the base network to a reduced one works as follows. The base network $\mathcal{B}$ is trained on a dataset D with a standard cross-entropy loss on each training sample, that is,

$$\mathcal{L}_{\text{CE}} = -\sum_{k=1}^{K} y_k \log\left(\frac{e^{\hat{p}_k}}{\sum_{j=1}^{K} e^{\hat{p}_j}}\right), \tag{4}$$

where K is the number of different classes, y represents the ground-truth one-hot vector of a sample, and $\hat{p}$ represents the output logits from the base network. The same loss is applied when training a hypothesis of reduced network $\mathcal{R}_\tau$, using its own output logits in place of $\hat{p}$.

Besides employing a cross-entropy loss, which maximizes the probability of the correct labels, we train each reduced network hypothesis to minimize a Kullback–Leibler divergence loss with respect to the output probabilities of the base network. Formally, this is defined as

$$\mathcal{L}_{\text{KL}} = -\sum_{k=1}^{K} \hat{z}_k \log\left(\frac{z_k}{\hat{z}_k}\right), \tag{5}$$

where $\hat{z}_k$ represents the soft targets from the base network $\mathcal{B}$, and z_k indicates the normalized output from the reduced network hypothesis. Formally,

$$\hat{z}_k = \frac{e^{\hat{p}_k/t}}{\sum_{j=1}^{K} e^{\hat{p}_j/t}} \quad \text{and} \quad z_k = \frac{e^{p_k/t}}{\sum_{j=1}^{K} e^{p_j/t}}, \tag{6}$$

where the same softmax temperature $t > 1$ is applied to both probability distributions. Following Reference 39, we also multiply the KL loss by t^2 when using both hard targets (corresponding to the ground-truth y adopted in Equation 4) and soft targets. The final objective used to train each reduced network hypothesis is a weighted sum of the cross-entropy and KL losses, as follows:

$$\mathcal{L}_S = \mathcal{L}_{\text{CE}} + \alpha t^2 \mathcal{L}_{\text{KL}}, \tag{7}$$

where α is a constant scalar determining the relative weight of the KL loss with respect to the cross-entropy loss.

Besides distilling knowledge from the base network by employing the activations from the last layer, in our preliminary experiments we also explored with different Knowledge Distillation methods involving intermediate feature maps—for example, by adopting a MSE loss or a partial L_2 distance together with a margin ReLU[41] between the Student's and the Teacher's activations, although without observing significant improvements. More details can be found in Section 4.5.

3.4 | Adaptively fine-tuning from previous iterations

So far, the reduced network obtained at a given iteration, $\mathcal{R}_\tau$, has been trained from scratch on the target dataset and by distilling knowledge from the base model $\mathcal{B}$. While this training strategy clearly creates a link between the current reduced network and the base model, we also aim at creating a training dependency with the reduced model obtained at the previous iteration, $\mathcal{R}_{\tau-1}$. Being the latter the best model reached so far in terms of the complexity/accuracy trade-off, we expect its knowledge to be beneficial for the network hypotheses which are developed at the next stage. To this aim, we implement an adaptive fine-tuning strategy that aims at recovering the knowledge from the reduced models obtained at previous iterations.

At each iteration τ, before training the reduced hypotheses $\epsilon_j(\mathcal{R}_{\tau-1})$, $\epsilon_j \in \{\epsilon_s, \epsilon_t, \epsilon_f, \epsilon_c, \epsilon_l\}$, we initialize them with the weights of the reduced network from the previous iteration $\mathcal{R}_{\tau-1}$, whenever this is feasible in terms of architectural constraints, that is, when all the weights of $\epsilon_j(\mathcal{R}_{\tau-1})$ and $\mathcal{R}_{\tau-1}$ have the same shape. Clearly, this is verified only when

applying an operator which impacts the input shape, that is, $\epsilon_s, \epsilon_t, \epsilon_f$. When a hypothesis requires to use an operator impacting the weights shape, we do not apply the fine-tuning strategy and train from scratch.

Our progressive shrinkage mechanism is presented in Algorithm 1, assuming that Equation (3) can be used in place of Equation (1). Note that the inner-loop of the algorithm can be parallelized, meaning that the exploration of the five reduced models at a given iteration can be performed at the same time, being the reduction operators independent of each other. For details about the load distribution over devices for a single training, please refer to Section 4.4.

Algorithm 1. Progressive architecture shrinkage

Data: Pre-trained base network $\mathcal{B}$, reduction operations $\{\epsilon_s, \epsilon_t, \epsilon_f, \epsilon_c, \epsilon_l\}$, number of iterations I;
Result: Reduced networks $\mathcal{R}_\tau$, $\tau \in \{1, ..., I\}$;
$\mathcal{R}_0 \leftarrow \mathcal{B}$;
for $\tau := 1 \rightarrow I$ **do**
  TopAcc = 0;
  for ϵ_j *in* $\{\epsilon_s, \epsilon_t, \epsilon_f, \epsilon_c, \epsilon_l\}$ **do**
    if ϵ_j *in* $\{\epsilon_s, \epsilon_t, \epsilon_f\}$ **then**
      $\mathcal{R}_\tau^j \leftarrow \epsilon_j(\mathcal{R}_{\tau-1})$;
      Initialize $\mathcal{R}_\tau^j$with weights from $\mathcal{R}_{\tau-1}$;
      Train $\mathcal{R}_\tau^j$with $\mathcal{L}_S$;
    else
      $\mathcal{R}_\tau^j \leftarrow \epsilon_j(\mathcal{R}_{\tau-1})$;
      Initialize $\mathcal{R}_\tau^j$with random weights;
      Train $\mathcal{R}_\tau^j$with $\mathcal{L}_S$;
    end
    if $Acc(\mathcal{R}_\tau^j) > TopAcc$ **then**
      $\epsilon_\tau^* \leftarrow \epsilon_j$;
      $\mathcal{R}_\tau \leftarrow \mathcal{R}_\tau^j$;
      TopAcc $\leftarrow$ Acc$(\mathcal{R}_\tau^j)$;
    end
  end
end

4 | EXPERIMENTAL RESULTS

4.1 | Datasets

We adopt the *Kinetics-400*[1] dataset for training all the reduced models obtained with our architecture shrinkage approach. Kinetics-400 consists of approximately 240k training and 20k validation videos belonging to 400 different human action classes, with each class comprising at least 400 videos. Following a common procedure in literature, we report both the top-1 and top-5 classification accuracy on the validation set as a metric of effectiveness, and the number of FLOPs as a metric for computational cost. Since the standard practice for inference consists in predicting class-probabilities for multiple spatiotemporal crops of the same video, and then averaging them to obtain the overall video prediction, we underline that the computational cost linearly increases with the number of crops adopted during inference.

We also evaluate the transfer capabilities of our reduced models by fine-tuning them on *UCF101*[9] and *HMDB51*.[10] UCF101 consists of about 13k videos belonging to 101 different action classes, while HMDB51 has only about 6k videos split across 51 classes. They both provide three different splits for training and testing, and we report the average accuracy over these splits. In our preliminary experiments, we also tested our progressive architecture shrinkage technique when training from scratch on UCF101 and HMDB51: although our strategy still improved the computational performance, we observed strong overfitting. More details can be found in Section 4.5.

4.2 | Implementation details

While in principle our algorithm could be applied to any video backbone, we here consider two recent spatiotemporal CNNs to showcase the effectiveness of our approach. Namely, we employ our PyTorch reimplementation of R(2+1)D-18[6] and SlowFast-4 × 16-R50[7] for all our experiments.

R(2+1)D-18. This backbone decomposes 3D convolutions into 2D spatial convolutions followed by 1D temporal ones. We adopt the 18-layers version of this backbone and train a base model $\mathcal{B}$ following the original implementation presented in Tran et al.:[6] during training, we resize video frames to 128 × 171 and apply random crop with size 112 × 112. Each input clip of the base network consists of 16 consecutive frames. In the reduced models, when downsampling the spatial size, we also downsample the crop size accordingly.

During each training epoch, we sample three clips per video from random temporal locations for temporal jittering. Synchronized stochastic gradient descent is adopted on 64 NVIDIA GPUs, with a total mini-batch size of 1536 (24 per GPU). The base learning rate is set to 0.96, with linear warm-up during the first 10 epochs. Afterward, the learning rate is divided by 10 every 10 epochs, in both the base model and reduced models. When training a reduced model starting from pretrained weights, that is, when applying the adaptive fine-tuning presented in Section 3.4, the base learning rate is instead divided by 10. Training is always completed in 45 epochs. During inference, we use 112 × 112 center crops from 10 clips uniformly sampled from the video. Output probabilities of these 10 clips are averaged to obtain video-level prediction.

SlowFast-4 × 16-R50.SlowFast networks consist of two pathways, a Slow path operating at low frame-rate, and a Fast one operating at higher frame-rate and employing fewer channels in convolutional layers. In this article, we consider a base SlowFast instantiation with a ResNet-50 backbone.[51] During the training of the SlowFast base network, the shorter side of the input video is resized to a random value in the interval [256, 320] and keeping the aspect ratio, then 224 × 224 clips are randomly cropped from the video. The raw clips length is set to 64 frames, and the Slow path samples four frames with stride 16 from each clip, while the Fast path samples 32 frames with stride 2 from each clip.[7] In reduced models, we downsample input and temporal sizes accordingly.

As with R(2+1)D, 3 clips are randomly sampled from a video for an epoch. Stochastic gradient descent on 128 GPUs is adopted, with a total mini-batch size of 2048 (16 per GPU). The base learning rate is set to 1.6, with linear warm-up during the first five epochs and cosine annealing after. Also in this case, the base learning rate is divided by 10 when training a reduced model with adaptive fine-tuning. Again, 45 epochs are performed on both the base and reduced models to lighten the computational requirements, while the original SlowFast implementation[7] suggested 256 epochs without temporal jittering. In inference, 256 × 256 center crops are extracted from 10 uniformly sampled clips (instead of 30[7]), and their softmax scores are averaged to obtain video-level prediction.

Other implementation details. The t value in Equation (6) is set to 5, while the α value in Equation (7) is 500. We fix the maximum number of iterations of the coordinate descent to 5. In order to roughly halve FLOPs, we set $\gamma_s = 1.4$, $\gamma_t = 2$, $\gamma_f = 2$, $\gamma_c = 1.4$, and $\gamma_l = 2$. For γ_l, we uniformly halve the layers in each residual block. For both R(2+1)D-18 and SlowFast-4 × 16-R50, we used a momentum of 0.9 and a weight decay of 10^{-4}. A dropout of 0.5 has been applied before the final classification layer when using the SlowFast backbone. Finally, since SlowFast consists of two paths which sample frames differently from the input video, we clarify how we handle the two sampling strategies when reducing the temporal resolution or the frame rate. Specifically, we first apply the chosen reduction operations to the input clip and then allow each path to sample frames from the reduced input according to its sampling strategy.

4.3 | Progressive Architecture Shrinkage evaluation

Results using the R(2+1)D-18 backbone. We start presenting the results obtained using the R(2+1)D-18[6] backbone on the Kinetics-400 dataset. The upper side of Table 1 reports the top-1 and top-5 accuracy of the reduced models obtained by our progressive architecture shrinkage approach, as well as the number of GFLOPs required by each architecture to process a single view, multiplied by the number of views adopted during inference for a given video. In the table, "R(2+1)D-18-PAS-*x*" indicates the reduced model obtained at iteration *x*. As it can be observed, the total drop in accuracy from the base network (R(2+1)D-18) to the reduced model obtained in the last iteration (R(2+1)D-18-PAS-5) is **5.6%** (from 69.2 to 63.6), while the number of GFLOPs required to process each view is reduced from **40.8** to **2.5**. It is also worth noting that the reduced model obtained at the first iteration (R(2+1)D-18-PAS-1) has a slightly increased accuracy compared with the base network (69.5 vs. 69.2), despite requiring half of its GFLOPs per view (40.8 vs. 20.4).

TABLE 1 Top-1, top-5 accuracy and inference cost (GFLOPs per view × number of adopted views) of progressively reduced models obtained from our PAS algorithm, starting from a R(2+1)D-18 and a SlowFast-4 × 16-R50 backbone

Model	Top-1	Top-5	GFLOPs
R(2+1)D-18[6]	69.2	88.1	40.8 × 10
R(2+1)D-18-PAS-1	69.5	88.7	20.4 × 10
R(2+1)D-18-PAS-2	68.8	88.1	10.2 × 10
R(2+1)D-18-PAS-3	67.5	87.0	5.4 × 10
R(2+1)D-18-PAS-4	66.0	86.3	3.2 × 10
R(2+1)D-18-PAS-5	63.6	84.6	2.5 × 10
SlowFast-4×16-R50[7]	73.5	91.3	36.6 × 10
SlowFast-4×16-R50-PAS-1	73.8	91.7	18.9 × 10
SlowFast-4×16-R50-PAS-2	73.4	91.1	10.2 × 10
SlowFast-4×16-R50-PAS-3	71.7	90.3	5.1 × 10
SlowFast-4×16-R50-PAS-4	69.8	88.9	2.9 × 10
SlowFast-4×16-R50-PAS-5	67.0	87.3	1.4 × 10

Note: Results are reported on the Kinetics-400 validation set.

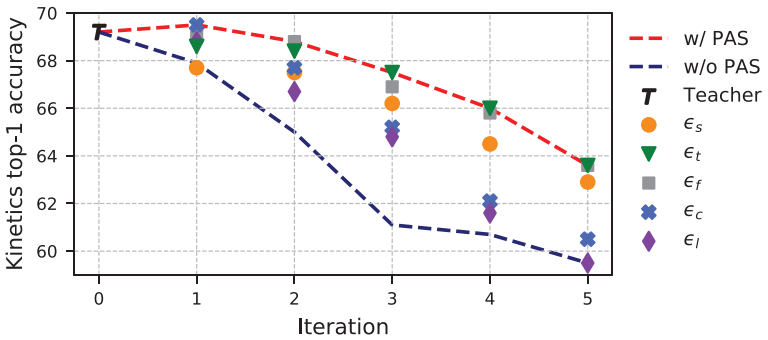

FIGURE 3 Top-1 accuracy of the PAS strategy applied to a R(2+1)D-18 backbone over five iterations. Different markers represent the accuracy of different reduction operations. The red-dashed line follows the sequence of best-performing models, while the blue one shows what happens if the best model is trained without using the Knowledge Distillation and the adaptive fine-tuning strategy

Figure 3 shows the detail of the reduction procedure and the accuracy of all the hypotheses obtained at each iteration. The red-dashed line highlights the best-performing hypothesis selected at each iteration. Specifically, the sequence of reduction operations which has been chosen by the coordinate descent is the following: $[\epsilon_c, \epsilon_f, \epsilon_t, \epsilon_t, \epsilon_t]$ – so the procedure firstly selects to reduce the number of channels, then the frame rate of the input clip, and then its temporal length (three times). Noticeably, different reductions may lead to the same top-1 accuracy, as it happens at iteration 5 for ϵ_t and ϵ_f: to choose the best operation in these cases, we sort the hypotheses using top-5 accuracy. The accuracy obtained by the other hypotheses, on which the remaining reductions were applied, is also reported at each step.

The blue-dashed line in Figure 3 represents, at each iteration, the accuracy of a model with the same architecture as the best-performing hypothesis, but trained without using the Knowledge Distillation and the adaptive fine-tuning strategy. This is further detailed in the top part of Table 2, where we report the advantage of progressively reducing R(2+1)D-18 models through our PAS algorithm, compared with a standard KD-free and cross-entropy based training from scratch. At the third iteration, for instance, after the reductions $[\epsilon_c, \epsilon_f, \epsilon_t]$, the proposed PAS provides a gain of 6.4% top-1 accuracy (67.5 vs. 61.1).

TABLE 2 Accuracy comparison between models trained with or without our Knowledge Distillation and adaptive fine-tuning, on the Kinetics-400 validation set

Model	With PAS		Without PAS	
	Top-1	Top-5	Top-1	Top-5
R(2+1)D-18-PAS-1	69.5	88.7	67.9	87.5
R(2+1)D-18-PAS-2	68.8	88.1	65.0	85.4
R(2+1)D-18-PAS-3	67.5	87.0	61.1	83.1
R(2+1)D-18-PAS-4	66.0	86.3	60.7	82.5
R(2+1)D-18-PAS-5	63.6	84.6	59.5	81.3
SlowFast-4×16-R50-PAS-1	73.8	91.7	73.1	91.0
SlowFast-4×16-R50-PAS-2	73.4	91.1	70.5	89.5
SlowFast-4×16-R50-PAS-3	71.7	90.3	68.1	88.4
SlowFast-4×16-R50-PAS-4	69.8	88.9	66.0	86.8
SlowFast-4×16-R50-PAS-5	67.0	87.3	61.2	83.6

Results using the SlowFast-4×16-R50 backbone. We also present results using the more recent SlowFast-4×16-R50[7] network. The lower side of Table 1 shows the accuracy and GFLOPs of a progressively reduced SlowFast instantiation using our PAS strategy. Accuracy drops from 73.5 (base network) to 67 (SlowFast-4×16-R50-PAS-5) in the sequence of iterations, while the number of GFLOPs per view is reduced from 36.6 to 1.4. Again, SlowFast-4×16-R50-PAS-1 improves the accuracy of the base model from 73.5 to 73.8, while cutting down the computational complexity from 36.6 to 18.9 GFLOPs per view.

Figure 4 depicts the PAS procedure for this backbone, using the same notation as in Figure 3. The best sequence of reduction operations here is $[\epsilon_c, \epsilon_s, \epsilon_t, \epsilon_s, \epsilon_f]$. Finally, in Table 2 bottom, we show the comparison between PAS-based SlowFast networks and their PAS-free counterparts at each iteration of our algorithm, as already observed for R(2+1)D-18. At iteration 5, the gap reaches its maximum value, with an absolute top-1 accuracy gain of 5.8%.

Observations. It is important to underline how the sequence of reduction operations chosen by our PAS method reflects which axes should be reduced in order to lower the computational requirements without compromising accuracy: for R(2+1)D-18, four out of five reductions involve time-related axes. This means that, for the given initial input shape ($16 \times 3 \times 112 \times 112$) defined by the authors,[6] the temporal dimension is the most redundant. SlowFast networks rely on stronger temporal modeling: ϵ_t and ϵ_f are indeed chosen only once, respectively. The input spatial resolution of 256×256 has some redundancy, and ϵ_s is chosen twice. Moreover, Figures 3 and 4 share a common trend: going on with iterations, ϵ_c and ϵ_l degrade performance much more compared with ϵ_s, ϵ_t, and ϵ_f. This gap confirms the usefulness of loading weights from previous iterations for input-related reductions, as explained in Section 3.4.

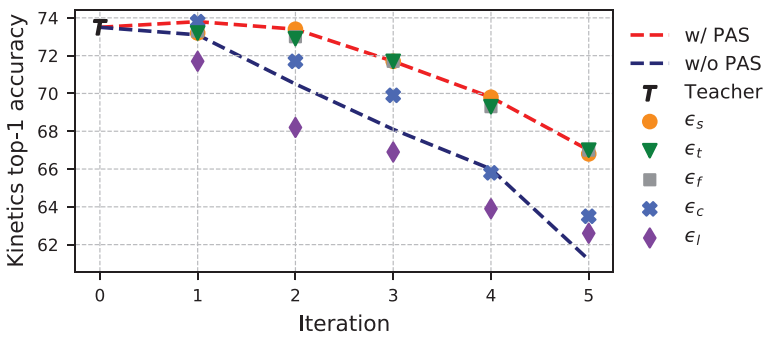

FIGURE 4 Top-1 accuracy of the PAS strategy applied to a SlowFast-4×16-R50 backbone over five iterations. Different markers represent the accuracy of different reduction operations. The red-dashed line follows the sequence of best-performing models, while the blue one shows what happens if the best model is trained without using the Knowledge Distillation and the adaptive fine-tuning strategy

TABLE 3 Comparison with existing action recognition models on the Kinetics-400 validation set, in terms of top-1 and top-5 accuracy

Model	Pretrain	Top-1	Top-5	GFLOPs
I3D[17]	ImNet	71.1	90.3	108 × N/A
Two-stream I3D[17]	ImNet	75.7	92.0	216 × N/A
Nonlocal R101[28]	ImNet	77.7	93.3	359 × 30
Nonlocal R50[28]	ImNet	76.5	92.6	282 × 30
TSM R50[5]	ImNet	74.7	N/A	65.0 × 10
MF-Net[35]	ImNet	72.8	90.4	11.1 × 50
Two-stream I3D[17]	None	71.6	90.0	216 × N/A
ip-CSN-152[32]	None	77.8	92.8	109 × 30
Oct-I3D+NL[52]	None	75.7	N/A	28.9 × 30
R(2+1)D-18[6]	None	69.2	88.1	40.8 × 10
SlowFast-4×16-R50[7]	None	73.5	91.3	36.6 × 10
R(2+1)D-18-PAS-1	None	**69.5**	**88.7**	**20.4 × 10**
SlowFast-4×16-R50-PAS-1	None	**73.8**	**91.7**	**18.9 × 10**

Note: Inference cost is also reported (GFLOPs per view × number of adopted views). PAS is bounded by the accuracy of the base network, but its effectiveness is confirmed on two different well-known backbones.
Abbreviation: ImNet, ImageNet.

TABLE 4 Accuracy obtained when removing Knowledge Distillation or fine-tuning from previous iteration

Model	KD	Fine-tuning	ϵ_s	ϵ_t	ϵ_f
R(2+1)D-18-PAS-3	✓	✓	**66.2**	**67.5**	**66.9**
R(2+1)D-18-PAS-3	×	✓	65.6	66.3	65.4
R(2+1)D-18-PAS-3	✓	×	64.5	65.7	65.3

Comparison with the state of the art. In order to directly compare the models obtained in the succession of PAS iterations with other well-known models, we present the performance of several existing methods on the Kinetics-400 validation set in Table 3. For each method, the top-1 and top-5 accuracy are shown, along with their pretraining strategy and inference time. Differently from X3D models,[8] which exploit an expansion algorithm that starts from a predefined architecture, the accuracy of our models is bounded by the base network capabilities: in this regard, PAS differs from other strategies which aims to maximize accuracy. We believe that the main advantage of the PAS algorithm consists in its generalization ability: one can choose an existing strong network (based on some requirements) and apply PAS on it to reduce its computational load while maintaining accuracy almost unaltered. As the ability to limit the accuracy loss is confirmed on two different backbones, with different designs and performance, PAS could potentially be applied to X3D models as well.

Ablation analysis. As mentioned, PAS leverages two key techniques to avoid an accuracy decay when progressively reducing models, that is, Knowledge Distillation and fine-tuning from previous iterations. In order to quantify the role of these two strategies, in Table 4 we present an ablation experiment conducted on iteration 3 of the R(2+1)D-18 backbone. Here we first remove the Knowledge Distillation from the Student training process, but we load pretrained weights from R(2+1)D-18-PAS-2. Then, we restore KD and train another Student from scratch, that is, with randomly initialized weights. In both cases, accuracy drops compared with our full solution. We explore only ϵ_s, ϵ_t, and ϵ_f reductions for ablation purposes, since fine-tuning is unfeasible when applying ϵ_c and ϵ_l, while applying KD but not fine-tuning is exactly what we propose for ϵ_c and ϵ_l.

Transfer capabilities to smaller datasets. We assess the transfer capabilities of PAS-generated models trained on the Kinetics-400 dataset, by fine-tuning them on UCF101 and HMDB51. When fine-tuning a model, the base learning rate

is divided by 10, while all the other implementation details remain the same as presented in Section 4.2. Noticeably, PAS strategy is not adopted again in these experiments: we simply fine-tune PAS models trained on Kinetics-400, to verify if the transfer capabilities are maintained in the sequence of PAS iterations.

Table 5 shows the top-1 classification accuracy when fine-tuning all the models obtained with our PAS strategy. As it can be seen, the transfer capability is preserved for both datasets: R(2+1)D-18-PAS-1 exceeds the accuracy of the base network on HMDB51, while SlowFast-4×16-R50-PAS-1 and SlowFast-4×16-R50-PAS-2 exceed the accuracy of the base network on UCF101, despite being much lighter. The first three SlowFast models reported in Table 5 have not been evaluated on HMDB51, since many videos last between 1 and 2 s, which makes unfeasible to sample 64 frames at a frame-rate of 30 fps. Being the reduction sequence for SlowFast $[\epsilon_c, \epsilon_s, \epsilon_t, \epsilon_s, \epsilon_f]$, the input temporal resolution is reduced to 32 at the third PAS iteration, which allows us to report the accuracy of the last three models on HMDB51.

4.4 | Computational analysis

Resource utilization during inference. Here we investigate the advantage of performing inference with a model shrinked by PAS, in terms of both GPU utilization and memory consumption. For fairness, we employ 4 NVIDIA V100 GPUs with 32 GB of memory, and collect statistics from each GPU over the whole Kinetics-400 validation set, using a batch size of 1. When measuring the average percentage utilization of CUDA cores and the average percentage of GPU memory consumption, we notice that they both linearly decrease, together with the number of FLOPs, when starting from a base model and going on with PAS iterations. Specifically, we measured a 39.1% and 11.3% GPU utilization and memory consumption, respectively, when using a base R(2+1)D-18. These values are reduced to 7.7% and 1.0%, respectively, when using R(2+1)D-18-PAS-5. A similar trend has been observed for SlowFast-4×16-R50 models.

Computational strategies at training time. Our algorithm employs significant computational resources to find the best sequence of reduction operations. Nevertheless, after exploring different network hypotheses, it provides a compressed model with a complexity/accuracy trade-off of choice. In this section, we report some quantitative details about the number of GPU-hours required to perform five iterations with PAS. With the implementation details presented in Section 4.2, training each network hypothesis requires about 15–16 h for both R(2+1)D-18 and SlowFast-4×16-R50. Using 64 GPUs for a single R(2+1)D-18 experiment, we employed ∼25.000 GPU-hours for training 25 network hypotheses (5 iterations × 5 reduction operators). For SlowFast-4 × 16-R50 training, we used 128 synchronized GPUs and employed a total number of ∼50.000 GPU-hours.

In all our experiments we used distributed training on nodes with 4 NVIDIA V100 GPUs each, distributing the computation across 16 and 32 nodes, respectively, when employing the R(2+1)D-18 backbone and the SlowFast-4 × 16-R50 backbone. With distributed training the model is replicated on each device and the input is partitioned along the batch

TABLE 5 Transfer ability of PAS models on UCF101 and HMDB51 datasets. K400 stands for Kinetics-400

Model	Pretrain	UCF	HMDB	GFLOPs
R(2+1)D-18[6]	K400	**94.4**	70.0	40.8 × 10
R(2+1)D-18-PAS-1	K400	94.0	**71.0**	20.4 × 10
R(2+1)D-18-PAS-2	K400	93.5	68.1	10.2 × 10
R(2+1)D-18-PAS-3	K400	91.6	63.3	5.4 × 10
R(2+1)D-18-PAS-4	K400	89.8	58.9	3.2 × 10
R(2+1)D-18-PAS-5	K400	87.0	54.0	2.5 × 10
SlowFast-4×16-R50[7]	K400	95.0	–	36.6 × 10
SlowFast-4×16-R50-PAS-1	K400	**95.3**	–	18.9 × 10
SlowFast-4×16-R50-PAS-2	K400	95.1	–	10.2 × 10
SlowFast-4×16-R50-PAS-3	K400	94.0	70.4	5.1 × 10
SlowFast-4×16-R50-PAS-4	K400	92.8	68.3	2.9 × 10
SlowFast-4×16-R50-PAS-5	K400	90.4	63.3	1.4 × 10

TABLE 6 Accuracy when using different KD approaches on intermediate activations

Initial model	Red. op.	Logits only	+MSE	+mReLU[41]
R(2+1)D-18	ϵ_l	68.8	68.1	68.0

TABLE 7 Accuracy gain when setting the base model $\mathcal{B}$ in training mode with respect to evaluation mode.

			Reduction operator				
Initial model	$\mathcal{B}$ train	$\mathcal{B}$ eval	ϵ_s	ϵ_t	ϵ_f	ϵ_c	ϵ_l
R(2+1)D-18	✓		67.7	68.6	69.0	69.5	68.8
R(2+1)D-18		✓	67.5	68.4	68.8	69.0	68.2

dimension so that each chunk is handled by a different device. During the backward pass, gradients from each device are averaged. We spawn a number of processes equal to the number of available GPUs, each exclusively working on a single GPU, and we exploit the `DistributedDataParallel` utility of the PyTorch library[53] which automatically handles synchronization and communication primitives between processes using the NVIDIA Collective Communication Library (NCCL). All training and evaluation experiments have been performed on the CINECA Marconi100 cluster, which consists of 980 GPU-powered nodes and is ranked 11th in the top500* ranking of the 500 most powerful commercially available computer systems in the world.

4.5 | Additional experiments

Distilling knowledge from intermediate feature maps. In Table 6, we investigate the usage of a Knowledge Distillation loss employing the activations from intermediate layers. Specifically, we build an additional loss between the activations of the base network and of each network hypothesis, which is applied after each residual block of the backbone. We test the usage of both an L_2 loss and the margin ReLU-based approach presented by Heo et al.[41] As it can be observed, the introduction of an additional loss on intermediate layers does not increase the top-1 accuracy. As a consequence, in the final formulation of PAS we only employed network logits for Knowledge Distillation.

Role of batch normalization. An important factor to consider when distilling the knowledge from the base network $\mathcal{B}$ to reduced models is the role of batch-normalization. As reported by Heo et al.,[41] Batch-norm layers should behave in the same way in the teacher and in the student (i.e., they should be both in training mode or in evaluation mode). For this reason, our base model is set to training mode when computing its logits for Knowledge Distillation, since the students are in training mode as well. This ensures that the features from both the teacher and the student are normalized in the same way. Table 7 shows the advantage of distilling the knowledge from an R(2+1)D-18 base model in training mode with respect to evaluation mode. Top-1 accuracy on Kinetics-400 is reported for the first iteration of PAS and for all reduction operations.

Training PAS on smaller datasets without Kinetics-400 pretraining. As anticipated in Section 4.1, we also trained PAS on UCF101 without employing a Kinetics-400 pretraining. Table 8 shows the top-1 accuracy of a progressively reduced R(2+1)D-18 model. For simplicity, we only report the top-1 accuracy obtained on the first split of UCF101. The top-1 accuracy of the base model $\mathcal{B}$ is 62.3. For each PAS iteration, the accuracy obtained by applying each reduction operator is reported, along with the accuracy of the best reduced model trained from scratch without any Knowledge Distillation (last column).

In the first iteration, while the accuracy drop caused by ϵ_s, ϵ_t, and ϵ_f is comparable with that observed in Kinetics-400, the same does not hold for ϵ_c and ϵ_l. Specifically, using PAS for reducing the number of channels or the number of layers increases the top-1 accuracy by 5.0% and 3.2%, respectively. Looking at the second row of Table 8, last column, it is clear that the accuracy gain is not completely due to PAS: when reducing the number of channels (best-performing reduction operator in the first iteration) and training the reduced model without PAS, accuracy still increases with respect to the base model (64.6 vs.62.3), even with 50% fewer FLOPs. This highlights that the model capacity of R(2+1)D-18 is

*https://top500.org/.

TABLE 8 PAS applied on the UCF101 dataset with a R(2+1)D-18 backbone

	Reduction operator					
Initial model	ϵ_s	ϵ_t	ϵ_f	ϵ_c	ϵ_l	**Without PAS**
R(2+1)D-18	–	–	–	–	–	62.3
R(2+1)D-18-PAS-1	60.5	62.0	61.5	**67.3**	65.5	64.6
R(2+1)D-18-PAS-2	63.8	66.3	66.2	**66.7**	66.5	64.0
R(2+1)D-18-PAS-3	63.5	65.4	**65.8**	65.3	65.1	60.3
R(2+1)D-18-PAS-4	62.8	**65.3**	64.1	63.0	63.5	59.3
R(2+1)D-18-PAS-5	62.2	63.7	**63.9**	61.8	62.2	57.8

excessive for UCF101, which leads to strong overfitting. The same is visible in the following iteration. Despite overfitting, we notice that PAS still plays a key role in improving accuracywhile reducing the number of FLOPs. From the third iteration on, reduced models trained without PAS start to get worse, while PAS-based training still ensures a minimal accuracy loss.

SlowFast-4 × 16-R50 network architecture. For reference, in Table 9 we report the detailed architecture of the base network and of the reduced hypotheses used in all our experiments with the SlowFast-4 × 16-R50 backbone.

TABLE 9 Architecture of the base model and of reduced hypotheses based on SlowFast-4 × 16-R50, as a function of the total reduction factor applied on spatial resolution, temporal length, frame rate, number of channels and number of layers ($\overline{\gamma}_s, \overline{\gamma}_t, \overline{\gamma}_f, \overline{\gamma}_c, \overline{\gamma}_l$)

Stage	***Slow* pathway**	***Fast* pathway**	**Output sizes $T \times S^2$**
Raw clip	–	–	$64/(\overline{\gamma}_t\overline{\gamma}_f) \times (224/\overline{\gamma}_s)^2$
Data layer	Stride 16, 1^2	Stride 2, 1^2	*Slow* : $4/(\overline{\gamma}_t\overline{\gamma}_f) \times (224/\overline{\gamma}_s)^2$ *Fast* : $32/(\overline{\gamma}_t\overline{\gamma}_f) \times (224/\overline{\gamma}_s)^2$
Conv_1	$1 \times 7^2, 64/\overline{\gamma}_c$ Stride 1, 2^2	$5 \times 7^2, 8/\overline{\gamma}_c$ Stride 1, 2^2	*Slow* : $4/(\overline{\gamma}_t\overline{\gamma}_f) \times (112/\overline{\gamma}_s)^2$ *Fast* : $32/(\overline{\gamma}_t\overline{\gamma}_f) \times (112/\overline{\gamma}_s)^2$
Pool_1	1×3^2 max Stride 1, 2^2	1×3^2 max Stride 1, 2^2	*Slow* : $4/(\overline{\gamma}_t\overline{\gamma}_f) \times (56/\overline{\gamma}_s)^2$ *Fast* : $32/(\overline{\gamma}_t\overline{\gamma}_f) \times (56/\overline{\gamma}_s)^2$
Res_2	$\begin{bmatrix} 1 \times 1^2, 64/\overline{\gamma}_c \\ 1 \times 3^2, 64/\overline{\gamma}_c \\ 1 \times 1^2, 256/\overline{\gamma}_c \end{bmatrix} \times 3/\overline{\gamma}_l$	$\begin{bmatrix} 3 \times 1^2, 8/\overline{\gamma}_c \\ 1 \times 3^2, 8/\overline{\gamma}_c \\ 1 \times 1^2, 32/\overline{\gamma}_c \end{bmatrix} \times 3/\overline{\gamma}_l$	*Slow* : $4/(\overline{\gamma}_t\overline{\gamma}_f) \times (56/\overline{\gamma}_s)^2$ *Fast* : $32/(\overline{\gamma}_t\overline{\gamma}_f) \times (56/\overline{\gamma}_s)^2$
Res_3	$\begin{bmatrix} 1 \times 1^2, 128/\overline{\gamma}_c \\ 1 \times 3^2, 128/\overline{\gamma}_c \\ 1 \times 1^2, 512/\overline{\gamma}_c \end{bmatrix} \times 4/\overline{\gamma}_l$	$\begin{bmatrix} 3 \times 1^2, 16/\overline{\gamma}_c \\ 1 \times 3^2, 16/\overline{\gamma}_c \\ 1 \times 1^2, 64/\overline{\gamma}_c \end{bmatrix} \times 4/\overline{\gamma}_l$	*Slow* : $4/(\overline{\gamma}_t\overline{\gamma}_f) \times (28/\overline{\gamma}_s)^2$ *Fast* : $32/(\overline{\gamma}_t\overline{\gamma}_f) \times (28/\overline{\gamma}_s)^2$
Res_4	$\begin{bmatrix} 3 \times 1^2, 256/\overline{\gamma}_c \\ 1 \times 3^2, 256/\overline{\gamma}_c \\ 1 \times 1^2, 1024/\overline{\gamma}_c \end{bmatrix} \times 6/\overline{\gamma}_l$	$\begin{bmatrix} 3 \times 1^2, 32/\overline{\gamma}_c \\ 1 \times 3^2, 32/\overline{\gamma}_c \\ 1 \times 1^2, 128/\overline{\gamma}_c \end{bmatrix} \times 6/\overline{\gamma}_l$	*Slow* : $4/(\overline{\gamma}_t\overline{\gamma}_f) \times (14/\overline{\gamma}_s)^2$ *Fast* : $32/(\overline{\gamma}_t\overline{\gamma}_f) \times (14/\overline{\gamma}_s)^2$
Res_5	$\begin{bmatrix} 3 \times 1^2, 512/\overline{\gamma}_c \\ 1 \times 3^2, 512/\overline{\gamma}_c \\ 1 \times 1^2, 2048/\overline{\gamma}_c \end{bmatrix} \times 3/\overline{\gamma}_l$	$\begin{bmatrix} 3 \times 1^2, 64/\overline{\gamma}_c \\ 1 \times 3^2, 64/\overline{\gamma}_c \\ 1 \times 1^2, 256/\overline{\gamma}_c \end{bmatrix} \times 3/\overline{\gamma}_l$	*Slow* : $4/(\overline{\gamma}_t\overline{\gamma}_f) \times (7/\overline{\gamma}_s)^2$ *Fast* : $32/(\overline{\gamma}_t\overline{\gamma}_f) \times (7/\overline{\gamma}_s)^2$
Global average pool, fc			#Classes

Note: Kernels are denoted as {T × S^2, C} for temporal, spatial, and channel sizes, while strides as {temporal stride, spatial stride^2}. Convolutional residual blocks are represented in brackets. The speed ratio between the Fast and the Slow paths is 8, while the channel ratio is 1/8.

The architecture is reported as a function of the total reduction factors applied on spatial resolution, temporal length, frame rate, number of channels and layers (denoted as $\overline{\gamma}_s$, $\overline{\gamma}_t$, $\overline{\gamma}_f$, $\overline{\gamma}_c$, $\overline{\gamma}_l$) as a result of applying a sequence of reduction operators. The base network $\mathcal{B}$ corresponds to the configuration where all $\overline{\gamma}_*$ are equal to 1, and is identical to the one proposed by Feichtenhofer et al.[7] At each iteration τ, a chosen reduction operation ϵ_j can increase the corresponding reduction factor $\overline{\gamma}_j$ by a factor of γ_j (see Section 3.1), and modifies the network architecture according to Table 9.

5 | CONCLUSION

In this article we presented PAS, a Progressive Architecture Shrinking approach which can iteratively reduce the computational demands of a video architecture while limiting the loss in accuracy. The proposed approach is based on a coordinate descent schema which aims at finding the best sequence of reduction operators to be applied on a base network. At each stage we maintain the knowledge learned in the base network and in previous iterations through a distillation and an adaptive fine-tuning strategy. The approach is implemented by exploring different network hypothesis in parallel, through the usage of an HPC cluster.

Experimental evaluations have been conducted on the Marconi100 cluster, employing two video prediction backbones, namely, R(2+1)D-18 and SlowFast-4 × 16-R50 and Kinetics-400, UCF101 and HMDB51 as datasets. Employing PAS on a R(2+1)D-18 allows to reduce the number of required GFLOPs from 40.8 to 2.5, while limiting the loss in accuracy on from 69.2 to 63.6 (Kinetics-400). When using a SlowFast-4 × 16-R50 backbone, instead, PAS reduces the number of required GFLOPs from 36.6 to 1.4, dropping accuracy from 73.5 to 67. Also, PAS does not alter the transfer learning capabilities of the base network toward small-scale datasets like UCF101 and HMDB51. As an additional contribution, we conducted tests on distilling knowledge from intermediate feature maps, on the role of Batch Normalization and on the usage of PAS on smaller scale datasets, avoiding Kinetics-400 pretraining.

ACKNOWLEDGMENT

Open Access Funding provided by Universita degli Studi di Modena e Reggio Emilia within the CRUI-CARE Agreement. [Correction added on 24 May 2022, after first online publication: CRUI funding statement has been added.]

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Matteo Tomei https://orcid.org/0000-0002-1385-924X

REFERENCES

1. Kay W, Carreira J, Simonyan K, et al. The kinetics human action video dataset; 2017. arXiv preprint arXiv:1705.06950.
2. Tran D, Bourdev L, Fergus R, Torresani L, Paluri M. Learning spatiotemporal features with 3D convolutional networks. Paper presented at: Proceedings of the International Conference on Computer Vision, Santiago, Chile; 2015.
3. Simonyan K, Zisserman A. Two-stream convolutional networks for action recognition in videos. Paper presented at: Proceedings of Advances in Neural Information Processing Systems, Montréal, Canada; 2014.
4. Xie S, Sun C, Huang J, Tu Z, Murphy K. Rethinking spatiotemporal feature learning: speed-accuracy trade-offs in video classification. Paper presented at: Proceedings of the European Conference on Computer Vision, Munich, Germany; 2018.
5. Lin J, Gan C, Han S. Tsm: temporal shift module for efficient video understanding. Paper presented at: Proceedings of the International Conference on Computer Vision, Seoul, Korea; 2019.
6. Tran D, Wang H, Torresani L, Ray J, LeCun Y, Paluri M. A closer look at spatiotemporal convolutions for action recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT; 2018.
7. Feichtenhofer C, Fan H, Malik J, He K. Slowfast networks for video recognition. Paper presented at: Proceedings of the International Conference on Computer Vision, Long Beach, CA; 2019.
8. Feichtenhofer C. X3D: expanding architectures for efficient video recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Virtual; 2020:203-213.
9. Soomro K, Zamir AR, Shah M. UCF101: a dataset of 101 human actions classes from videos in the wild; 2012. arXiv preprint arXiv:1212.0402.
10. Kuehne H, Jhuang H, Garrote E, Poggio T, Serre T. HMDB: a large video database for human motion recognition. Proceedings of the International Conference on Computer Vision; 2011.
11. Köpüklü O, Kose N, Gunduz A, Rigoll G. Resource efficient 3D convolutional neural networks. Paper presented at: Proceedings of the International Conference on Computer Vision Workshops, Seoul, Korea; 2019.
12. Xiao L, Meng Y, Tian X, Luo H. Energy allocation for activity recognition in wearable devices with kinetic energy harvesting. *Softw Pract Exper*. 2021;1–18.

13. Sankar H, Subramaniyaswamy V, Vijayakumar V, Arun Kumar S, Logesh R, Umamakeswari A. Intelligent sentiment analysis approach using edge computing-based deep learning technique. *Softw Pract Exper*. 2020;50(5):645-657.
14. Guedria S, De Palma N, Renard F, Vuillerme N. R2D2: a scalable deep learning toolkit for medical imaging segmentation. *Softw Pract Exper*. 2020;50(10):1966-1985.
15. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. Imagenet: a large-scale hierarchical image database. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Miami, FL; 2009.
16. Hara K, Kataoka H, Satoh Y. Can spatiotemporal 3D CNNS retrace the history of 2D CNNS and imagenet? Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT; 2018.
17. Carreira J, Zisserman A. Quo vadis, action recognition? a new model and the kinetics dataset. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Honolulu, HI; 2017.
18. Varol G, Laptev I, Schmid C. Long-term temporal convolutions for action recognition. *IEEE Trans Pattern Anal Mach Intell*. 2017;40(6):1510-1517.
19. Feichtenhofer C, Pinz A, Wildes R. Spatiotemporal residual networks for video action recognition. Paper presented at: Proceedings of Advances in Neural Information Processing Systems, Barcelona, Spain; 2016.
20. Luo C, Yuille AL. Grouped spatial-temporal aggregation for efficient action recognition. Paper presented at: Proceedings of the International Conference on Computer Vision, Seoul, Korea; 2019.
21. Qiu Z, Yao T, Mei T. Learning spatio-temporal representation with pseudo-3D residual networks. Paper presented at: Proceedings of the International Conference on Computer Vision, Venice, Italy; 2017.
22. Horn BK, Schunck BG. Determining optical flow. *Artif Intell*. 1981;17(1-3):185-203.
23. Feichtenhofer C, Pinz A, Zisserman A. Convolutional two-stream network fusion for video action recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Las Vegas, NV; 2016.
24. Wang L, Xiong Y, Wang Z, et al. Temporal segment networks: towards good practices for deep action recognition. Paper presented at: Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands; 2016.
25. Hussein N, Gavves E, Smeulders AW. Timeception for complex action recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA; 2019.
26. Li C, Zhong Q, Xie D, Pu S. Collaborative spatiotemporal feature learning for video action recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA; 2019.
27. Li X, Shuai B, Tighe J. Directional temporal modeling for action recognition. Paper presented at: Proceedings of the European Conference on Computer Vision, Virtual; 2020.
28. Wang X, Girshick R, Gupta A, He K. Non-local neural networks. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT; 2018.
29. Wang X, Xiong X, Neumann M, et al. AttentionNAS: spatiotemporal attention cell search for video classification. Paper presented at: Proceedings of the European Conference on Computer Vision, Virtual; 2020.
30. Wang Z, Lin C, Sheng L, Yan J, Shao J. PV-NAS: practical neural architecture search for video recognition; 2020. arXiv preprint arXiv:2011.00826.
31. Meng Y, Lin CC, Panda R, et al. AR-Net: adaptive frame resolution for efficient action recognition. Paper presented at: Proceedings of the European Conference on Computer Vision, Virtual; 2020.
32. Tran D, Wang H, Torresani L, Feiszli M. Video classification with channel-separated convolutional networks. Paper presented at: Proceedings of the International Conference on Computer Vision, Seoul, Korea; 2019.
33. Bilen H, Fernando B, Gavves E, Vedaldi A, Gould S. Dynamic image networks for action recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Las Vegas, NV; 2016.
34. Carreira J, Patraucean V, Mazare L, Zisserman A, Osindero S. Massively parallel video networks. Paper presented at: Proceedings of the European Conference on Computer Vision, Munich, Germany; 2018.
35. Chen Y, Kalantidis Y, Li J, Yan S, Feng J. Multi-fiber networks for video recognition. Paper presented at: Proceedings of the European Conference on Computer Vision, Munich, Germany; 2018.
36. Diba A, Fayyaz M, Sharma V, et al. Spatio-temporal channel correlation networks for action classification. Paper presented at: Proceedings of the European Conference on Computer Vision, Munich, Germany; 2018.
37. Wu CY, Zaheer M, Hu H, Manmatha R, Smola AJ, Krähenbühl P. Compressed video action recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT; 2018.
38. Zolfaghari M, Singh K, Brox T. Eco: efficient convolutional network for online video understanding. Paper presented at: Proceedings of the European Conference on Computer Vision, Munich, Germany; 2018.
39. Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network. Proceedings of the Paper presented at: Advances in Neural Information Processing Systems Workshops, Montréal, Canada; 2015.
40. Wang L, Yoon KJ. Knowledge distillation and student-teacher learning for visual intelligence: a review and new outlooks; 2020. arXiv preprint arXiv:2004.05937.
41. Heo B, Kim J, Yun S, Park H, Kwak N, Choi JY. A comprehensive overhaul of feature distillation. Paper presented at: Proceedings of the International Conference on Computer Vision, Seoul, Korea; 2019.
42. Kim K, Ji B, Yoon D, Hwang S. Self-knowledge distillation: a simple way for better generalization; 2020. arXiv preprint arXiv:2006.12000.
43. Mirzadeh SI, Farajtabar M, Li A, Levine N, Matsukawa A, Ghasemzadeh H. Improved knowledge distillation via teacher assistant. Paper presented at: Proceedings of the Conference on Articial Intelligence, New York, NY; 2020.

44. Yue K, Deng J, Zhou F. Matching guided distillation. Paper presented at: Proceedings of the European Conference on Computer Vision, Virtual; 2020.
45. Yuan L, Tay FE, Li G, Wang T, Feng J. Revisiting knowledge distillation via label smoothing regularization. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Virtual; 2020.
46. Bhardwaj S, Srinivasan M, Khapra MM. Efficient video classification using fewer frames. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA; 2019.
47. Thoker FM, Gall J. Cross-modal knowledge distillation for action recognition. Paper presented at: Proceedings of the International Conference on Image Processing, Taipei, Taiwan; 2019.
48. Garcia NC, Morerio P, Murino V. Modality distillation with multiple stream networks for action recognition. Paper presented at: Proceedings of the European Conference on Computer Vision, Munich, Germany; 2018.
49. Garcia NC, Bargal SA, Ablavsky V, Morerio P, Murino V, Sclaroff S. DMCL: distillation multiple choice learning for multimodal action recognition; 2019. arXiv preprint arXiv:1912.10982.
50. Wright SJ. Coordinate descent algorithms. *Math Program*. 2015;151(1):3-34.
51. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Las Vegas, NV; 2016.
52. Chen Y, Fan H, Xu B, et al. Drop an octave: reducing spatial redundancy in convolutional neural networks with octave convolution. Paper presented at: Proceedings of the International Conference on Computer Vision, Seoul, Korea; 2019.
53. Paszke A, Gross S, Massa F. Pytorch: an imperative style, high-performance deep learning library. Paper presented at: Proceedings of Advances in Neural Information Processing Systems, Vancouver, Canada; 2019.

AUTHOR BIOGRAPHIES

Matteo Tomei received the master's degree in Computer Engineering in 2018, and he is currently a Ph.D. student at the AImageLab, the artificial intelligence and computer vision laboratory of the University of Modena and Reggio Emilia, in Italy. His work focuses on Multimedia and Deep Learning technologies for video understanding. His research involves spatiotemporal action detection, efficient video models, privacy-preserving approaches for video understanding, and deep learning for Soccer analysis and sports. He also worked on generative networks in the field of cultural heritage and art.

Lorenzo Baraldi received the Ph.D. degree cum laude in Information and Communication Technologies from the University of Modena and Reggio Emilia, Italy, in 2018. He is currently Assistant Professor at the "Enzo Ferrari" Engineering Department of the University of Modena and Reggio Emilia, Italy. He was a Research Intern at Facebook AI Research (FAIR) in 2017. He has authored or coauthored more than 60 publications in scientific journals and international conference proceedings. His research interests include image processing, video understanding, deep learning and multimedia.

Giuseppe Fiameni is a Data Scientist at NVIDIA where he oversees the NVIDIA AI Technology Center in Italy, a collaboration among NVIDIA, CINI and CINECA to accelerate academic research in the field of Artificial Intelligence through collaboration projects. He has been working as HPC specialist at CINECA, the largest HPC facility in Italy, for more than 14 years providing support for large-scale data analytics workloads. Research interests include large-scale deep learning models, system architectures, massive data engineering, video action detection and anticipation.

Simone Bronzin has a background in physics and more than 20 years of experience in IT industry. During his career he focused on applying machine and deep learning methods to develop innovative solutions that brought real value to enterprises across different industries. In 2016 he founded Metaliquid, a computer vision startup that aims to change the way media companies manage and produce video contents. He is currently Metaliquid CEO.

Rita Cucchiara received the Ph.D. degree in Computer Engineering from the University of Bologna, Bologna, Italy, in 1992. She is currently a Full Professor of computer engineering with the University of Modena and Reggio Emilia, Modena, Italy. She is the Director of the national CINI Laboratory on Artificial Intelligence and Intelligent Systems, the head of the Modena Unit of the ELLIS Society, and director of the AImageLab Laboratory at the University of Modena and Reggio Emilia. She has authored or coauthored more than 400 papers in journals and international proceedings, and is a reviewer for several international journals. She has been a coordinator of several projects in computer vision and pattern recognition, and in particular on video surveillance, human behavior analysis, video understanding, human-centered searching in images and video, and big data for cultural heritage.

How to cite this article: Tomei M, Baraldi L, Fiameni G, Bronzin S, Cucchiara R. A computational approach for progressive architecture shrinkage in action recognition. *Softw Pract Exper*. 2022;52(2):537-554. doi: 10.1002/spe.3035