NAS just once: Neural Architecture Search for joint Image-Video Recognition

Supplementary Material

This supplementary material is intended to provide implementation details to allow procedure reproducibility. All experiments were run on 8 NVIDIA A100 GPU with 40 GB of memory. Searching and training took from 1 day for the smallest model to 5 days for the largest VIM-NAS-L model. Appendix A gives hyper-parameters details for the supernet training. Appendix B describes the pseudo-code for supernet training under a joint-datasets scenario and supernet selection through a perturbation-based differentiable approach. Finally, Appendix C overviews the evolutionary algorithm we deployed to select top-performing subnetworks under resource constraints.

A. Implementation details

We summarize the hyperparameters employed in our experiments in Tab. 10. We employ for all the datasets random spatial and temporal cropping. The provided details refer to single-dataset experiments, presented in the first column of Tab. 8b in the main paper. When joint training, the learning rate to be used is ImageNet's, while the batch size is the smallest among the video datasets. We experienced some training instability on the search-space related to larger ViT models, *i.e*. ViT-B and ViT-L. Therefore, we decreased the weight decay value and the learning rate, to avoid the drop of the training accuracy to 0 and the flat loss experienced with larger values.

In the joint training scenario, we use separate fully connected layers to output the correct class predictions alternatively selected given the sampled dataset, *e.g*., for ImageNet and Kinetics-600, we use two different FC layers, one with 1,000 and one with 600 outputs. The loss is computed independently for each relevant head and is backpropagated at it at each optimizer step. As previously stated in Sec. 3 of the main paper, gradients are not accumulated as we found the procedure to be detrimental to performance.

B. Details on Supernet Training

In this section we detail the elements of our supernet joint-training strategy. Algorithm 1 describes the process of supernet training with differentiable weight entanglement. The algorithm is divided into two steps: we train the supernet on ImageNet for epochs = $1, \dots E/3$, while from $E/3$ to the end of training all four datasets are alternately sampled according to a weight mechanism related to the number of training samples in the considered dataset. A different ViT sample α is randomly selected at each iteration. Its weights are obtained from the supernet's ones $\boldsymbol{w}_{\mathcal{A}}$ and and used to compute loss of the subnet $\mathcal{A}(\alpha; w)$. At last, we update the corresponding weights in $\boldsymbol{w}_{\mathcal{A}}$ while freezing the rest and updating the topological structure through the validation loss. The final architecture is selected through a perturbation-based approach, which differs from DARTS. Indeed, rather than choosing the maximum α values, we progressively remove layer choices associated with the α parameters and identify as best component the one leading to the highest validation accuracy drop.

Algorithm 1: Supernet Training

Input: Training epochs E, search space $\mathcal{S}$, supernet $\mathcal{A}$, initial supernet weights $\boldsymbol{w}_{\mathcal{A}}$, train datasets $\mathcal{D}_{i=1,\dots,4}$, Loss $\mathcal{L}$
Output: Optimal architecture
for $i = 1, \dots, E/3$ **do**
  for *data, targets in* $\mathcal{D}_1$ **do**
    Random sample one ViT architecture $\alpha = (\alpha^{(1)}; \dots \alpha^{(i)}; \dots \alpha^{(l)})$ from the space $\mathcal{S}$ where l is the maximum depth;
    Sample the related weights $\boldsymbol{w} = (w^{(1)}; \dots w^{(i)}; \dots w^{(l)})$ from $\boldsymbol{w}_{\mathcal{A}}$; Compute the gradients $\nabla_{\boldsymbol{w}}$ based on $\mathcal{L}$, data, and targets;
    Backpropagate the Loss;
    Update the corresponding part of $\boldsymbol{w}$ in $\boldsymbol{w}_{\mathcal{A}}$ while freezing the remaining part of $\mathcal{A}$;
    Update the topological parameters according to $\mathcal{A} = \mathcal{A} - \gamma \nabla_{\mathcal{A}} \mathcal{L}_{val}(\boldsymbol{w}^*, \mathcal{A})$;
for $i = E/3, \dots, E$ **do**
  for *each optimizer step* **do**
    Sample *data, targets* from $\mathcal{D}_1, \dots, \mathcal{D}_4$ according to $|\mathcal{D}|$;
    Random sample one ViT architecture α as in step 5;
    Obtain the corresponding weights $\boldsymbol{w} = (w^{(1)}; \dots w^{(i)}; \dots w^{(l)})$ from $\boldsymbol{w}_{\mathcal{A}}$;
    Compute the gradients $\nabla_{\boldsymbol{w}}$ based on $\mathcal{L}$, data, and targets;
    Backpropagate the Loss;
    Update the corresponding part of $\boldsymbol{w}$ in $\boldsymbol{w}_{\mathcal{A}}$ as in step 7;
    Update the topological parameters according to $\mathcal{A} = \mathcal{A} - \gamma \nabla_{\mathcal{A}} \mathcal{L}_{val}(\boldsymbol{w}^*, \mathcal{A})$;
return *selected* $\alpha^* \leftarrow \mathcal{A} \setminus \alpha$ *giving lowest val. accuracy drop*;

Table 10. Summary of hyperparemeters employed in our experiments. For Vit-B and Vit-L denoted as (L), a different learning rate and weight decay were used.

	ImageNet	K400	K600	SSV2
Optimization				
Optimizer		AdamW		
Batch size	1024	256	256	256
Learning rate schedule		cosine + linear warmup		
Linear warmup steps		20		
Base learning rate	1e-3 (L 1e-5)	1e-4 (B 5e-5, L 2e-5)		2e-5
Epochs		500		
Data augmentation				
Rand augment		✓		
Repeated augmentations		✗		
Random erasing		✓		
Weight Decay	5e-2	0.001 (L 1e-5)		1e-5
Cutmix	1.0	✗	✗	✗
Mixup	0.8	✗	✗	0.3
Drop path		0.1		
Label Smoothing	0.1	✗	✗	0.3
Number of Frames	1	64	64	32
FPS	-	15	15	24

C. Details on Evolutionary Algorithm

The evolutionary algorithm is used as an alternative search algorithm to find the optimal architectures when the search needs to be performed under resource constraints. Moreover with our setup, because the subnetworks are all well trained, the evolutionary algorithm brings the advantage of obtaining many subnetworks well-solving the task. Therefore, as in the differentiable formulation we did not place additional constraints on resources as we noticed it would have led to unstable training, we detail the evolutionary algorithm we deployed for the resource-constrain scenario. In Algorithm 2 crossover is realized by randomly selecting two candidate architectures from the top ones. A block is then uniformly chosen from each candidate in each layer to generate a new child network. Mutation verifies by changing a candidate depth with probability P_d and then mutating each block with a probability of P_m. As the evolution algorithm is deployed under resource constraints, a final check chooses to add the newly produced architectures based on the satisfaction of the selected constraint (*e.g.* the number of parameters).

Algorithm 2: Evolutionary Algorithm

Input: Search space $\mathcal{S}$, supernet $\mathcal{A}$, supernet weights $\boldsymbol{w}^*_{\mathcal{A}}$, population size N, resource constraints f, # of generation iterations $\mathcal{T}$, validation dataset $\mathcal{D}_{\text{val}}$, mutation probability of depth P_d, mutation probability of each layer P_m
Output: The best ViT α^* satisfying the contraints
$G_{(0)}$:= Random sample N architectures $(\alpha_1, \ldots, \alpha_N)$ from $\mathcal{A}$ with the constraint f;
while *search step* $t \in (0, \mathcal{T})$ **do**
  for $\alpha_i \in G_{(t)}$ **do**
    Get the related weight $\boldsymbol{w}_{\alpha_i}$ from the supernet weights $\boldsymbol{w}_{\mathcal{A}}$;
    Calculate the accuracy of the subnet $\mathcal{A}(\alpha_i, w_{\alpha_i})$ on the validation set $\mathcal{D}_{\text{val}}$;
G_{topk} := the Top K candidates by accuracy order;
$G_{\text{crossover}}$:= Crossover(G_{topk}, C);
G_{mutation} := Mutation($G_{\text{topk}}, P_d, P_m, C$);
$G_{(t+1)} = G_{\text{crossover}} \cup G_{\text{mutation}}$;
return *Best performing subnetworks*;