7th ICML Workshop on Automated Machine Learning (2020)

Tiny Video Networks: Architecture Search for Efficient Video Models

AJ Piergiovanni AJPIERGI@GOOGLE.COM
Google Research & Robotics at Google

Anelia Angelova ANELIA@GOOGLE.COM
Google Research & Robotics at Google

Michael S. Ryoo MRYOO@GOOGLE.COM
Google Research, Robotics at Google & Stony Brook University

Abstract

Video understanding is a challenging problem with great impact on real-world applications. Yet, solutions so far have been computationally intensive, with the fastest algorithms running at few hundred milliseconds per video snippet on powerful GPUs. We use architecture search to build highly efficient models for videos - Tiny Video Networks - which run at unprecedented speeds and, at the same time, are effective at video recognition tasks. The Tiny Video Networks run faster than real-time e.g., at less than 20 milliseconds per video on a GPU and are much faster than contemporary video models. These models not only provide new tools for real-time applications such as in mobile vision and robotics, but also enable fast research and development for video understanding. The project site is available at `https://sites.google.com/view/tinyvideonetworks`.

1. Introduction

Understanding videos is a crucial visual task. Successful methods for video analysis use complex and computationally intensive neural network models (Tran et al., 2014; Carreira and Zisserman, 2017; Xie et al., 2018; Wang et al., 2018). These approaches however are not suitable for real-time video processing, which greatly hinders their application to real-world systems, e.g., in robotics, or for mobile devices, where compute is limited.

Neural Architecture Search (Zoph and Le, 2017; Zoph et al., 2018; Liu et al., 2019; Real et al., 2019) has opened new avenues for creating both highly accurate and efficient models.

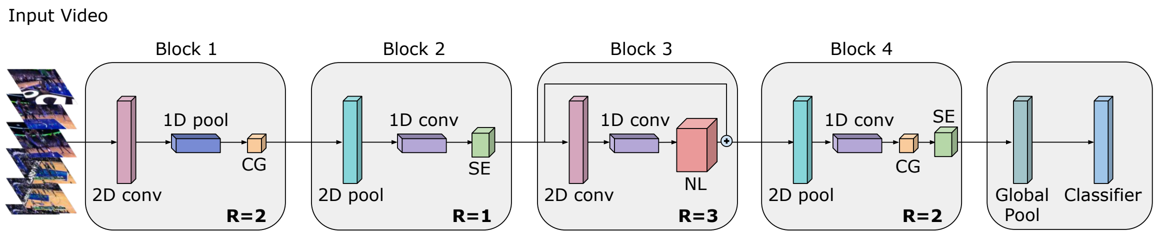

Figure 1: An example of a highly efficient 'Tiny Video Network', working on a video snippet. TVN-1 is shown. It takes 37 ms (CPU), 10ms (GPU).

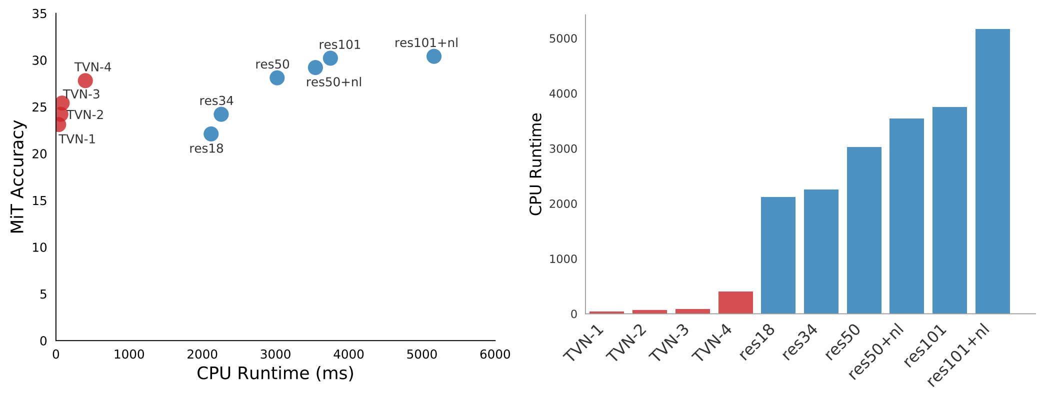

Figure 2: Runtime vs. model accuracy of the Tiny Video Networks compared to the main video recognition models. The Tiny Video Networks are faster-than-real-time models for videos and much faster than contemporary video models: e.g. TVN-1 is 100x faster than ResNet-101.

However, architecture search for videos presents additional computational challenges as video models need to parse spatio-temporal information across multiple frames.

In this work we use architecture search techniques to construct the fastest to date models for video understanding. More specifically, we automatically design 'tiny' neural networks for video understanding (Figure 1), which achieve competitive accuracy and run efficiently, at real-time or better speeds, within 10 to 19 ms on a GPU per video clip.[1] We call them Tiny Video Networks (TVN), as they require extremely small runtimes, which is unprecedented for video models. The discovered architectures offer new non-obvious layer combinations. Figure 2 shows that the Tiny Video Networks operate in the high accuracy and low runtime area of the curve where no other models exist.

We use an architecture search which is designed to address the challenges of working with videos and at the same time producing novel, fast and accurate video models. Our approach allows for configuring and expanding a set of components which enable efficient new layer combinations to capture both spatial and temporal information. We additionally adapt the TVN search, to create the first video models suitable for deployment on a mobile device. Furthermore, our approach allows for more exploration of video architectures at very low cost. This will also allow for future video architecture work to be much more efficient and less computationally burdensome. To our knowledge, Tiny Video Networks are the fastest standalone video networks known to date, which run on both CPU and GPU with better than real-time speeds.

Related Work. Designing computationally efficient networks has been important area of research (Wofk et al., 2019; Wu et al., 2019; Zhang et al., 2018; Xiong et al., 2019). Advances in neural architecture search for images (Zoph and Le, 2017; Real et al., 2017; Liu et al., 2019) demonstrated large gains in recognition accuracy but are also successful in automatically building time-constrained models (Howard et al., 2019; Tan et al., 2019;

1. A video snippet in most of the datasets considered here consists of 32 frames, which span 1 second; some datasets have longer video durations.

Pham et al., 2018; Yang et al., 2018; Wu et al., 2019). Architecture search for videos has been relatively scarce, with the exception of (Piergiovanni et al., 2019b; Ryoo et al., 2020). Online video understanding, which focuses on fast video processing by reusing computations across frames, e.g., ECO, TSM (Zolfaghari et al., 2018; Lin et al., 2019), is also related. Our approach is complementary as our efficient standalone video TVNs can be further utilized for even faster online recognition.

2. Tiny Video Networks

Our search method uses the tournament selection evolutionary algorithm with discrete mutation operators (Goldberg and Deb, 1991; Real et al., 2019), as it allows parallel evaluation and mutation of multiple individuals in the population (here network architectures). We explore building networks constrained for runtime and, optionally, the number of parameters. We search for the optimal combination of layers, e.g. number of layers, their type (e.g., convolutional, pooling) and their configurations (kernel size, stride, etc). With videos in mind, we explore various input resolution, both spatial (width and height) and temporal (how many frames to sample from the video), as well as, skip connections and nonlinearities. We also consider efficient layers, such as, 2D spatial or 1D temporal convolutional layer, 1D pooling, non-local blocks (Wang et al., 2018), context-gating layers (Miech et al., 2017), and squeeze-and-excitation layers (Hu et al., 2018), where the key is that combination of these can build powerful video models. As a result, we build a huge search space of possible architectures and use evolution architecture search for exploration. Evolution is advantageous as it explores effectively the very large space of video architectures and also the irregular search space with a non-differentiable objective function.

In order to learn novel efficient video architectures, we maximize the following equation. Let N be the network configuration, which is defined in the subsection below, and θ denote the learnable weights of the network ($|\theta|$ is the number of weights in the network), and P be a hyperparameter controlling the maximum size of the network. We denote by $\mathcal{R}(N_\theta)$ the function which computes the runtime of the network on a device, given the network N with its weight values θ, and by R the maximum desired computational runtime. We optimize:

$$\begin{aligned} \underset{N_\theta}{\text{maximize}} \quad & \mathcal{F}(N_\theta) \\ \text{subject to} \quad & \mathcal{R}(N_\theta) < R \\ & |\theta| < P, \end{aligned} \tag{1}$$

where $\mathcal{F}$ is the fitness function, which measures the accuracy of the trained model on the validation set of a dataset. The search starts with a pool of random architectures, here 200. Each random architecture is generated by a random sample from the search space of all possible architectures. For example, a network first randomly selects the input resolutions, number of frames, and etc. Then it selects a fixed number of blocks as a uniform random variable between 1 and 8, and number of 'repeats' per block (up to 8). Then, per each block, we randomly sample a sequence of layers.

After evaluating these networks, we apply tournament selection: from the current population of 200 networks, we randomly choose 50 of them and take the top performing network as a 'parent.' We then apply a discrete 'mutation' operation to this network by randomly changing one part of the network, e.g. randomly changing one of the input resolution, the number of blocks, or layer type. After mutation, the new network is tested for runtime and subsequently trained and evaluated. If its performance is adequate (we use Top1+Top5 accuracy), it is added to the current population and the lowest performing network is removed. Each model is trained for 10,000 iterations, and since they are fast, the average training time is about 1.5 hours. The search is done within several hours to a day. Networks which do not satisfy the runtime are discarded without evaluation, speeding up the search.

3. Experiments

We conduct experiments on four well-established public video datasets, **Moments-in-time (MiT)** (Monfort et al., 2018), **HMDB** (Kuehne et al., 2011), **MLB** (Piergiovanni and Ryoo, 2018), **Charades** (Sigurdsson et al., 2016) the latter two evaluating multi-class multi-label tasks, and containing longer videos. We use the established evaluation protocols for these datasets and report runtime (both on CPU and GPU), FLOPs and accuracy. Runtime is measured on an Intel Xeon CPU running at 2.9GHz and a single V100 GPU. Note that most SOTA methods are computationally expensive and prior work does not report runtimes. We follow the specified network and inputs for each model: we use 32 frames, as in (2+1)D ResNet (Tran et al., 2018); I3D (Carreira and Zisserman, 2017) and S3D (Xie et al., 2018) use more frames to report results (64). Our results use RGB-only as inputs. Additionally using optical flow is known to improve performance but computing flow is expensive needing more time than a TVN network inference itself. Models are evolved on multiple datasets, e.g. TVN-1 on MiT, TVN-2 on MLB, etc., and are cross-evaluated to test their usability across datasets. As baselines, we compare to (2+1)D ResNets (Tran et al., 2018; Wang et al., 2018), S3D (Xie et al., 2018), and I3D (Carreira and Zisserman, 2017).

Figure 2 shows the runtime vs the accuracy of TVNs, together with prior methods. We can see that TVN models are significantly faster than all others, and only outperformed by much bigger and slower models. We note that achieving such inference speeds is an impressive result for videos. Tables 1, 2, 3, and 4 show the performance of the Tiny Video Networks evaluated on the four datasets for video recognition. For all datasets, TVNs perform similarly to previous state-of-the-art methods at a fraction of the cost. For example, Tiny Video Networks, with 23.1 and 24.2 accuracy, both outperform ResNet-18 and are 57 and 33 times faster, respectively; they are at the same performance as ResNet-34, while being 61 and 35 times faster. TVN-1 is 100 times faster than the commonly used ResNet-101 model for videos (Table 1). TVN-4 is close in accuracy to (Wang et al., 2018) despite having 154 times fewer FLOPs (Table 3). TVNs have fewer GFLOPs compared to 38-245 for ResNet-18/101. We futher note that prior online video models (Zolfaghari et al., 2018; Lin et al., 2019) also have at least twice GFLOPs (32-65) than TVNs.

Ablations. We further scale up TVN-1 in all dimensions (input resolution, width and depth) and also using the scaling coefficients based on the findings of (Tan and Le, 2019). Our scaled up model achieves 28.2 % accuracy for 305ms on CPU, and is able to achieve comparable performance to much larger models, still being very efficient. Please see the extended version Piergiovanni et al. (2019a) and the Appendix for more ablation results.

Table 1: Results on the MiT dataset comparing different Tiny Video Networks to baselines and state-of-the-art. TVNs achieve similar performance at a fraction of the compute cost. They are RGB-only. No runtime was reported in prior works.

Method	Runtime (ms) (CPU/GPU)	GFlops	Accuracy
ResNet-18	2120 / 105	38	21.1%
ResNet-34	2256 / 110	50	24.2%
ResNet-50	3022 / 125	124	28.1%
ResNet-101	3750 / 140	245	30.2%
TSN (Wang et al., 2016)	-	-	24.1%
2D ResNet-50 (Monfort et al., 2018) (pretr.)	-	-	27.1%
I3D (Monfort et al., 2018) (RGB+Flow)	-	-	29.5%
TVN-1 (MiT)	37 / 10	13	23.1%
TVN-2 (MLB)	65 / 13	17	24.2%
TVN-3 (Charades)	85 / 16	69	25.4%
TVN-4 (MiT)	402 / 19	106	27.8%

Table 2: Performance on MLB. * Our measurement of runtime.

Method	Runtime (CPU)	Runtime (GPU)	mAP
InceptionV3	-	-	47.9
I3D (Piergiovanni and Ryoo, 2018)	1865ms*	-	48.3
I3D+sub-events (Piergiovanni and Ryoo, 2018)	-	-	55.5
TVN-1 (MiT)	37ms	10ms	44.2
TVN-2 (MLB)	65ms	13ms	48.2
TVN-3 (Charades)	85ms	16ms	46.5
TVN-4 (MiT)	402ms	19ms	52.3

Table 3: Performance on Charades. TVNs are only outperformed by much bigger models.

Method	Runtime (ms) (CPU/GPU)	GFlops	mAP
CoViAR, Res-50 (Wu et al., 2018)	-	-	21.9
Asyn-TF, VGG16 (Sigurdsson et al., 2017)	-	-	22.4
I3D	-	216	32.9
Nonlocal, R101(Wang et al., 2018)	-	544×30	37.5
TVN-1 (MiT)	37 / 10	13	32.2
TVN-2 (MLB)	65 / 13	17	32.5
TVN-3 (Charades)	85 / 16	69	33.5
TVN-4 (MiT)	402 / 19	106	35.4

Table 4: Performance on HMDB. Prior work does not report runtime.

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
I3D (Carreira and Zisserman, 2017)	-	-	74.8%
S3D-G (Xie et al., 2018)	-	-	75.9%
TVN-1 (MiT)	37ms	10ms	72.1%
TVN-2 (MLB)	65ms	13ms	73.5%
TVN-3 (Charades)	85ms	16ms	71.8%
TVN-4 (MiT)	402ms	19ms	74.7%

Table 5: TVN models obtained when expanding the search with MobileNet components.

Method	Runtime (CPU)	Params.	GFlops	Accuracy
TVN-1	37ms	11.1M	13.0	23.1%
TVN-1 + swish	39ms	11.1M	13.0	24.8%
TVN-M-1	43ms	5.6M	10.0	21.95%
TVN-M-2	75ms	5.4M	10.1	21.96%

Mobile-friendly Tiny Video Networks. We make a modification to our search space to include mobile-friendly components – inverted residual layers and the hard swish activation function, similar to MobileNet models (Sandler et al., 2018), applied both in space and time dimensions. Table 5 shows two selected mobile models, named TVN-M-1 and TVN-M-2. They are comparable to TVN-1, but are able to achieve 23% fewer Flops and have almost twice fewer parameters, which are both very important for mobile, at a small reduction in accuracy. Furthermore, we modified the original TVN-1 by substituting all ReLu activations with the hard-swish (Howard et al., 2019), we found an improvement in accuracy of 1.7% with only negligible (2ms) loss in runtime, confirming its usefulness.

Comparison to MobileNet models. We compare our TVNs to a MobileNetV3-equivalent ones, by training a video-adapted model from per-frame MobileNetV3 (Howard et al., 2019). Table 6 shows that TVNs are advantageous in both accuracy and speed, especially notable are the significant improvements for both, for larger number of frames.

Conclusion. We present novel, efficient 'tiny' neural networks for videos. They are automatically discovered, perform well, as seen on four datasets, and are many times faster than contemporary video models.

Table 6: TVNs outperform MobileNet when applied to videos, in both rutime and accuracy.

# Frames	MobileNet Runtime	MobileNet Acc.	TVN Rumtime	TVN Acc.
1 Frame	42ms	18.8%	32ms	20.2%
2 Frame	58ms	19.3%	37ms	23.1%
8 Frame	280ms	20.8%	85ms	25.4%

References

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017.

David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Vijay Vasudevan Ruoming Pang, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *CoRR:1905.02244*, 2019.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *CVPR*, 2018.

Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *ICCV*. IEEE, 2011.

Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding. In *iccv*, 2019.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture seach. In *ICLR*, 2019.

Antoine Miech, Ivan Laptev, and Josef Sivic. Learnable pooling with context gating for video classification. In *Youtube 8M CVPR Workshop*, 2017.

Mathew Monfort, Alex Andonian, Bolei Zhou, Kandan Ramakrishnan, Sarah Adel Bargal, Tom Yan, Lisa Brown, Quanfu Fan, Dan Gutfruend, Carl Vondrick, et al. Moments in time dataset: one million videos for event understanding. *arXiv preprint arXiv:1801.03150*, 2018.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

AJ Piergiovanni and Michael S. Ryoo. Fine-grained activity recognition in baseball videos. In *CVPR Workshop on Computer Vision in Sports*, 2018.

AJ Piergiovanni, Anelia Angelova, and Michael S Ryoo. Tiny video networks. In *CoRR:1910.06961*, 2019a.

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

Esteban Real, Sherry Moore, Andrew Selle, Yutaka Leon Suematsu Saurabh Saxena, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Michael S. Ryoo, AJ. Pierjivanni, Mingxing Tan, and Anelia Angelova. Assemblenet: Searching for multi-stream neural connectivity in video architectures. In *ICLR*, 2020.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, , and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision*, 2016.

Gunnar A Sigurdsson, Santosh Divvala, Ali Farhadi, and Abhinav Gupta. Asynchronous temporal fields for action recognition. In *CVPR*, 2017.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *CVPR*, 2019.

Du Tran, Lubomir D Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3d: generic features for video analysis. *CoRR, abs/1412.0767*, 2(7):8, 2014.

Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, pages 6450–6459, 2018.

Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision*, pages 20–36. Springer, 2016.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, pages 7794–7803, 2018.

Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. Fastdepth: Fast monocular depth estimation on embedded systems. In *International Conference on Robotics and Automation*, 2019.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. In *CVPR*, pages 6026–6035, 2018.

Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *European Conference on Computer Vision*, pages 305–321, 2018.

Yunyang Xiong, Ronak Mehta, and Vikas Singh. Resource constrained neural network architecture search: Will a submodularity assumption help? In *ICCV*, 2019.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandlerand, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural networkadaptation for mobile applications. In *European Conference on Computer Vision*, 2018.

X. Zhang, X. and Zhou, M. Lin, and J. Sun. Shufflenet:an extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

Mohammadreza Zolfaghari, Kamaljeet Singh, and Thomas Brox. Eco: Efficient convolutional network for online video understanding. In *European Conference on Computer Vision*, 2018.

Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

Appendix A. Appendix

A.1 Additional visualizations

Figure 3 visualizes the remaining networks used in the paper TVN-2, TVN-3, TVN-4. Code will be released where detailed specification of all networks is provided.

Figure 4 visualizes one of the networks built with mobile components (here, TVN-M-1, on MiT). The inv-bottleneck is the inverse bottleneck layer used by MobileNet (Sandler et al., 2018), which generally saves parameters without harming performance.

For completeness we also visualize the single-frame network used in some of our ablation experiments (Figure 5).

A.2 Found TVN Models

We here describe the found Tiny Video Networks (TVNs), each one from learning with different constraints. TVN-1 is the fastest model found. It was found by constraining the search space to include models only running in less than 50ms on CPU (it runs at 37ms and was evolved on Moments-in-Time). TVN-2 is found by limiting the search space to 100ms and 12 million parameters (it runs at 65ms and was evolved on MLB). TVN-3 was found by limiting the search space to 100ms as well, but no constraint on the number of parameters. It runs at 85 ms and was evolved on Charades. Finally, TVN-4 is a slower model, found by allowing networks up to 1200ms and 30 million parameters (a max computation cost roughly comparable to I3D). It runs at 402ms on CPU and is evolved on Moments-in-Time. These models also have picked specific runtime image resolutions, number of frames f, and stride s: TVN1:224x224, f=2, s=4; TVN2:256x256, f=2, s=7; TVN-3:160x160, f=8, s=2; TVN-4: 128x128, f=8, s=4.

A.3 Search space

We here provide additional details about the evolution process.

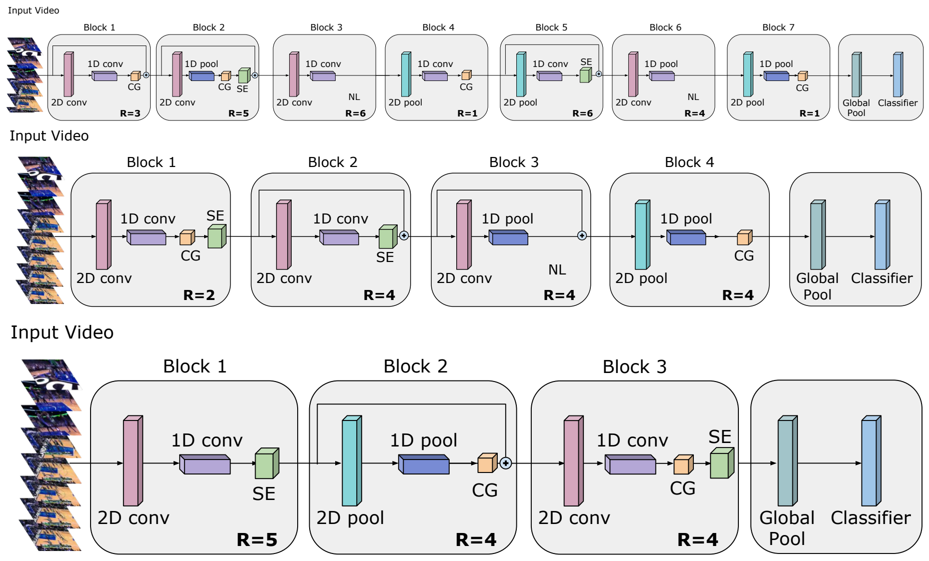

Figure 3: Example Tiny Video Networks found using architecture evolution showing several blocks with different configurations. A Tiny Video Net has multiple blocks, each repeated R times. Each block has a different configuration with spatial and temporal convolution, pooling, non-local layers, context gating and squeeze-excitation layers. It can select the image resolution and frame rate. From top to bottom: TVN-2, TVN-3, TVN-4. TVN-1 is shown in Figure 1.

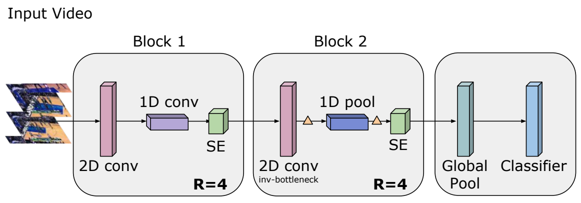

Figure 4: Illustration of TVN-Mobile. The triangle indicates a hard-swish activation function (no triangle is standard ReLU). Note that each block is repeated 4 times.

In order to generate a random network to start the evolution, one can sample from each of the components of the search space. For example, a network first randomly selects the input resolution which can be between (32×32 to 320×320) with a step size of 32. Then it will randomly pick the number of frames (1-128), and framerate - 1fps to 25fps (this is also referred to as 'stride', i.e. number of frames to skip; the stride is selected by uniform random sampling but depends on the number of frames already selected). Then it selects a fixed number of blocks as a uniform random variable between 1 and 8, and number of 'repeats' per block (up to 8). Then, per each block, we randomly sample a sequence of layers. They are selected randomly from a potential set of components, which are: 2D spatial or 1D temporal convolutional layer, 1D pooling, non-local blocks Wang et al. (2018),

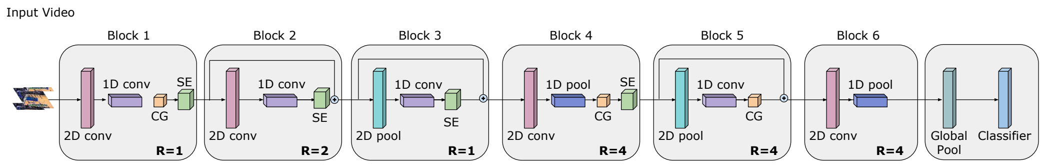

Figure 5: Illustration a 1-frame TVN. It uses many layers and repeats as it does not have to spend much compute on the temporal aspect.

context-gating layers Miech et al. (2017), and squeeze-and-excitation layers Hu et al. (2018). A residual connection at the end of a block can also be (randomly) enabled. Blocks are used for simplicity only and are not required. Their use reduces the search space somewhat, as the structure imposed by the blocks, eliminates some combinations. We found that this is still a very effective and little-constraint search space.

For each of these layers, a specific set of parameters are also sampled, in order to fully form a computational layer. For non-local layers, we search for the bottleneck size (between 4 and 1024). We search for the squeeze ratio for the squeeze-and-excitation layers (a real-valued number between 0 and 1). The convolutional layers can have a variety of kernel sizes (from 1 to 8), strides (from 1 to 8), number of filters (from 32 to 2048) and types (e.g., standard convolution, depthwise convolution, average pooling or max pooling). Additionally, a layer can optionally pick an activation function, a ReLu (or a swish for the Mobile-friendly models). The final block is followed by a fixed standard block of global average pooling, a dropout layer (0.5 fixed dropout rate), and a fully-connected layer which outputs the number of classes required for classification.

Since we are working with videos, exploring all of these potential architectures leads to a very large search space. Each block has $\sim 2^{34}$ possible configurations. When including the input resolution and up to 8 blocks in the network, the search space has a size of $\sim 2^{45}$ or about $\sim 10^{13}$. Thus, without automated search (e.g., if we do a brute-force grid search or random search), finding good architectures in this space is prohibitive. We here use such a large and unconstrained search space to find the optimal use of temporal and spatial information within a specific computational budget.

A.4 Ablations

A.4.1 Exploring a range of number of frames

We found that for some datasets (e.g., Moments-in-time and HMDB), the network prefers to use very few frames (e.g., 2 or 4 frames) to reduce the computation cost which is natural, given that runtime is the only constraint. Further, on these datasets, many activities are scene-based (e.g., swimming and baseball appear very differently), so a single frame is often enough to discriminate them. For example, our search was able to produce a reasonably well performing 1-frame model on MiT, with accuracy of 20.2%, 32ms runtime, 8 GFlops, at 224x224 resolution (Figure 5).

To determine the effect of temporal information on performance, we increased the number of inputs frames used by TVN-1 from 2 to 8 and 16, and re-trained these models on

Table 7: Increasing the number of frames for TVN-1 from 2 to 16 on MiT. We find that just adding more frames as input is not greatly beneficial.

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
TVN-1 (2 frames)	37ms	10ms	23.1%
TVN-1 (8 frames)	140ms	28ms	23.4%
TVN-1 (16 frames)	200ms	45ms	23.5%

Table 8: Different methods of scaling up the model on MiT. We explore scaling up spatial resolution (res), width, depth.

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
TVN-1	37ms	10ms	23.1%
TVN-1 (2x res)	140ms	28ms	23.5%
TVN-1 (4x res)	200ms	45ms	24.1%
TVN-1 (2x wide)	130ms	38ms	23.8%
TVN-1 (4x wide)	275ms	60ms	24.2%
TVN-1 (2x deep)	181ms	44ms	23.7%
TVN-1 (4x deep)	270ms	65ms	23.9%

MiT, providing more input information to the model. The results are shown in Table 7. We find that increasing the number of frames for TVN-1 on MiT does not lead to significant performance increase, while the runtime increases a lot, which explains the choice of selecting few frames.

A.4.2 Scaling Up the TVNs

We further demonstrate the performance of the models by scaling up the found Tiny Video Networks. In Table 8, we compare TVN-1 with increasing spatial resolution, increasing the width (number of filters in each layer) and increasing the depth (number of times each block is repeated). We simply scale these by multiplying them by 2 or 4. We find that scaling resolution and width lead to performance gains, but also that they may not be the most effective ones.

Based on the findings of EfficientNet (Tan and Le, 2019), we further scale up TVN-1 in all dimensions (input resolution, width and depth) using coefficients as in (Tan and Le, 2019). Table 9 shows the results. Our scaled up model, denoted as TVN-1 EN, is able to achieve comparable performance to much larger models, still being very efficient. This is an interesting result as is also an easy way of generating not-so-small but still very fast

Table 9: Scaling up our tiniest model (TVN-1) on MiT based on EfficientNet coefficients (denoted as TVN-1 EN).

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
(2+1)D ResNet-50	3022ms	125ms	28.1%
TVN-1	37ms	10ms	23.1%
TVN-1 EN	305ms	92ms	28.2%

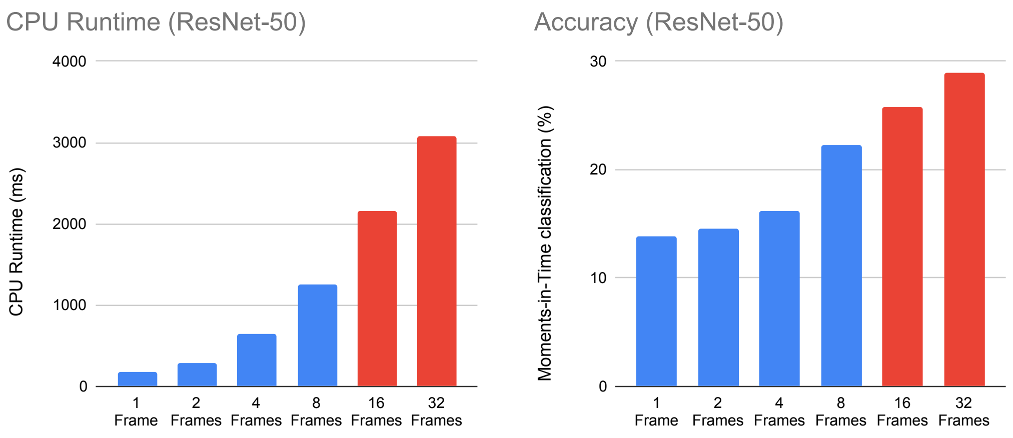

Figure 6: Comparing different number of frames using a standard (2+1)D ResNet-50. The last two bars (drawn in red) are for models with accuracy higher than the accuracy of TVN-1. As seen, even using 1-frame (2+1)D ResNet-50 has 185ms runtime on CPU, far slower than a TVN, with much poorer accuracy. More accurate (2+1)D ResNet-50 are much slower with runtime of more than 2000ms on CPU.

models. Conceivably, one can further evolve these models, and obtain even better accuracy at a fraction of the speed.

A.4.3 Comparison to (2+1)D ResNet-50

We further compare TVNs to standard ResNet-50 models for the same number of frames. In Figure 6, we show performance of (2+1)D ResNet-50 with varying number of input frames. Even using 1-frame, a ResNet-50 has 185ms runtime on a CPU, far slower than a TVN. Further, the accuracy of TVNs outperforms ResNet-50 until 16 frames are used. At that point, the runtime is over 2000ms, making it impractical for real-time mobile devices. In conclusion, contemporary models are slower and less accurate than TVNs for video tasks, even in the large number of frames setting.

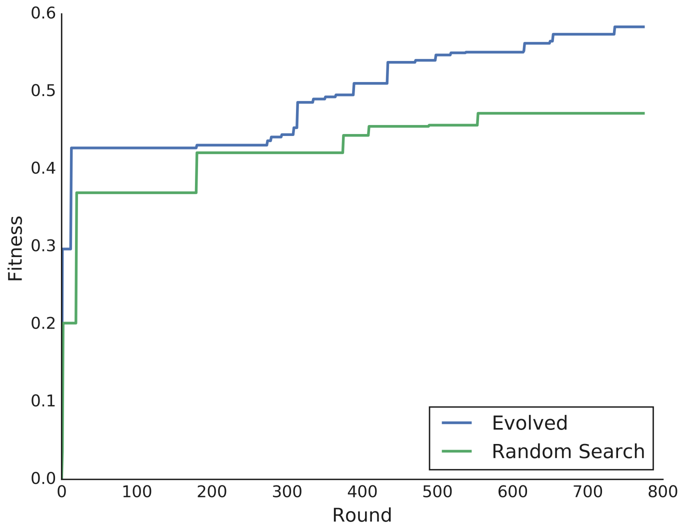

Figure 7: Random search vs. evolution for the TVNs search space. Evolution yields better models more quickly.

A.5 Evolution

We also observe that the evolution itself is beneficial compared to random search and produces much better models. Figure 7 shows the fitness for our evolved models vs random search.