Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

LETTER

Tiny Video Networks

AJ Piergiovanni*[1] | Anelia Angelova[1] | Michael S. Ryoo[1,2]

[1]Google Research, Robotics at Google, California, USA
[2]Dept. of Computer Science, Stony Brook University, New York, USA

Correspondence
*Corresponding author. Email: ajpiergi@google.com, anelia@google.com, mryoo@google.com

Summary

Automatic video understanding is becoming more important for applications where real-time performance is crucial and compute is limited. Yet, accurate solutions so far have been computationally intensive. We propose efficient models for videos - Tiny Video Networks - which are video architectures, automatically designed to comply with fast runtimes and, at the same time are effective at video recognition tasks. The Tiny Video Networks run at faster-than-real-time speeds and demonstrate strong performance across several video benchmarks. These models not only provide new tools for real-time video applications, but also enable fast research and development in video understanding. Code and models are available.

KEYWORDS:
efficient video models, video architecture search, video understanding

1 | INTRODUCTION

Video information is ubiquitous and abundant: media- and user-generated videos, data acquired by mobile devices or robots, are all sources of videos. Understanding videos is an important problem in computer vision with many applications, e.g. automated video tagging, activity recognition, robot perception. Video understanding tasks (action classification or detection, Figure 1) are challenging as they have the inherent complexity of image understanding, and additionally need to incorporate spatio-temporal information across multiple frames. Previous methods for video analysis use complex and computationally intensive models [1,2,3]. These approaches are not suitable for real-time video processing, which greatly hinders their application to real-world systems.

To address these challenges, we propose to automatically design video networks that provide strong recognition performance at a fraction of the computational cost. More specifically, we propose a general video architecture search approach, based on evolution, which designs a family of 'tiny' neural networks for video understanding. The networks achieve high accuracy and run efficiently, outperforming state-of-the-art video models, despite being order of magnitude faster (Figure 2). They run at real-time or better speeds, e.g. less than 20 ms on a GPU per video snippet[1]. We call them Tiny Video Networks (TVN), as they require extremely small runtimes, which is unprecedented for standalone video models. TVNs operate in the high accuracy and low runtime (or GFlops) area of the accuracy-runtime curve where no other models exist (Figure 2).

Video architecture search is a challenging task as the search needs to span the full spatio-temporal domain, while processing an order of magnitude more data per example. Neural architecture search approaches [4,5,6] are successful for image understanding, but are time-intensive even for images. With TVNs we address both problems in video architecture search, showing that it is possible to design efficient video architectures differently, both structurally and at the component level, from low-level primitives.

The contributions of this work are: (i) Creating highly efficient, faster-than-real-time Tiny Video Networks, with very strong performance on video understanding tasks, despite working at the fraction of the cost. (ii) We discover novel and interesting video architectures, which are easy to understand and can be easily integrated in computationally demanding video understanding tasks. Figure 3 shows example discovered architectures. (iii) We propose a search space which enables the combination of efficient layers, which effectively interleave components working in the time and spatial domains. As a result, the architecture search is

[1]A video snippet typically consists of 32 frames, which span 1 second; for longer video durations, the same video snippets are given to TVNs and baselines.

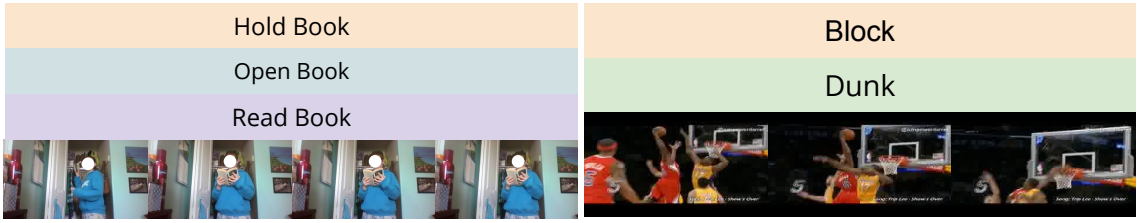

FIGURE 1 Examples of video understanding tasks: one or more action labels are provided per video snippet. The goal is to automatically recognize the action(s) per video. Some tasks include short video snippets and single classification label per video, some are long video clips with multiple (unordered) labels per video i.e. multi-class multi-label task. We use both settings here.

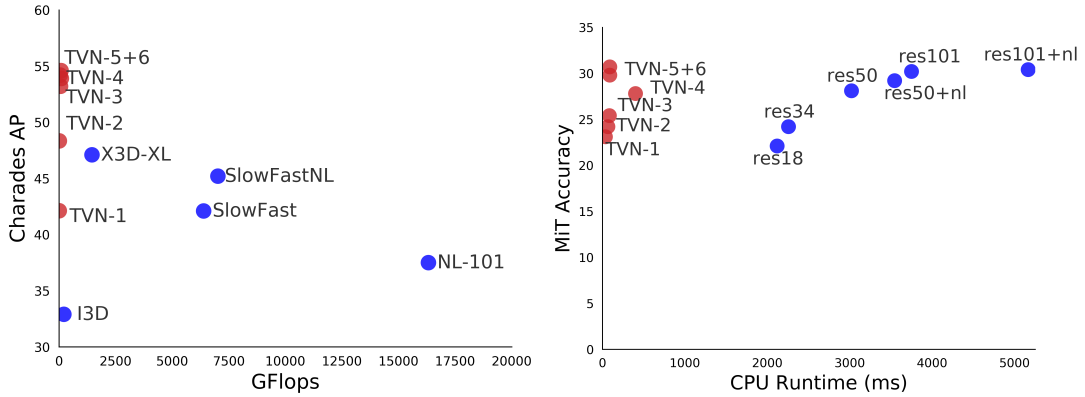

FIGURE 2 GFlops vs average precision (AP) of Tiny Video Networks (TVN) compared to the state-of-the-art, Charades dataset (left). Runtime vs. model accuracy of TVNs compared to the 3D-ResNet family models, MiT dataset (right).

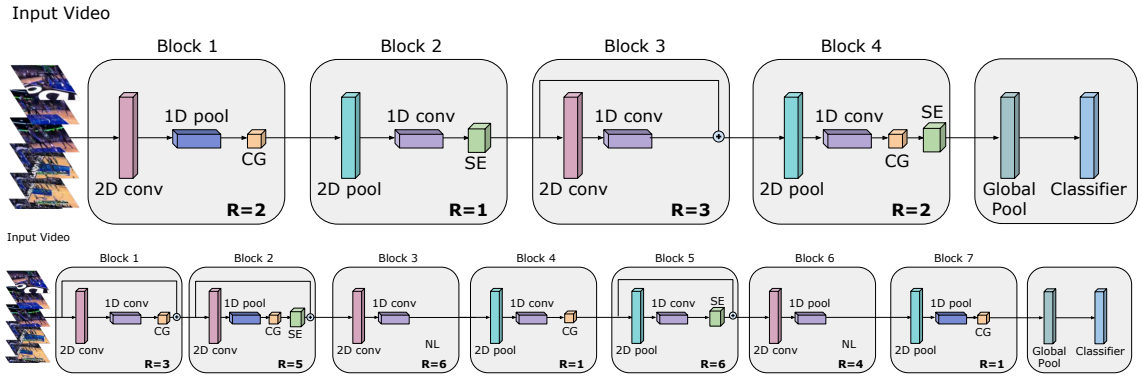

FIGURE 3 Example efficient Tiny Video Networks (TVN), created using architecture search. TVN-1 (top) takes 37 ms on CPU (10ms on GPU) for inference. TVN-2 (bottom) takes 65ms on CPU and 13ms on GPU. Each net has multiple blocks, each repeated *R* times. Each block has a different configuration with spatial and temporal convolution, pooling, non-local layers, context gating and etc. The architecture search can also select the input resolution, number of frames to sample and frame rate.

very efficient, despite being done on videos. (iv) Lastly, using the Tiny Video Networks architectures, we create TVNs suitable for mobile devices. Our Mobile TVNs significantly outperform the popular MobileNets[7] applied to videos, both in accuracy and runtime. We evaluate TVNs on well-established video understanding tasks for multi-class and multi-label multi-class action classification and test their generalizability on four challenging datasets: Moments-in-Time[8], HMDB[9], Charades[10], MLB[11].

TVNs can find wide application in real-time video understanding tasks, such as mobile phones or robots. Furthermore, they can be used for tasks which require heavy computational loads, e.g. server-side video processing, or for tasks that can benefit from time and energy savings. Due to their fast speeds, TVNs can also impact video research, by allowing researchers to train and explore future video architectures at very low cost thus accelerating video understanding research.

Code for the Tiny Video Network is open-sourced[2]. This paper is an extended version of the workshop contribution[12].

[2]Code: https://github.com/google-research/google-research/tree/master/tiny_video_nets. Trained models are publicly available via TF.Hub e.g. at https://tfhub.dev/google/tiny_video_net/tvn1/1, https://tfhub.dev/google/tiny_video_net/tvn2/2, etc.

2 | RELATED WORK

Traditionally, building efficient networks has been an important problem with a wealth of research, mostly limited to the image domain[13,14,7,15,16,17,18]. Network architectures have also been designed for specific hardware or mobile devices, e.g.,[13,7,15,16,17,19,20], where larger networks are optimized to run at fast speeds on these devices.

Video understanding research has produced a number of successful approaches [21,22,2,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,1,39,40,41,42,43,44], a handful of these have focused on efficient video processing[44,33].

Advances in neural architecture search (NAS)[4,45,6,46,47] demonstrate large gains in recognition accuracy and successful results. NAS methods can also be used for building automatically time-constrained models[13,15,5,48], in some cases with hardware in the loop[49,17]. Unlike architecture search for images[4,45,6,46,47], architecture search for videos has been scarce[50,51], primarily because of the computational cost of the architecture search itself. These approaches had produced relatively expensive networks. Recent work, X3D, generates efficient video models[52], where the networks are configured by expanding fast 2D models. Unlike prior work in video architecture search, we here create efficient video architectures automatically from first principles which are both highly efficient and accurate, with competitive performance to the best architecture-searched models[51].

Some works reduce the computation cost of video CNNs. Representation flow[53], MFNet[23] reduce the computation of optical flow, while CoViAR[54] focus on using compressed videos (e.g., MPEGs) to perform recognition. These works rely on heavy CNNs to obtain strong results. We here focus on obtaining efficient and compact CNN architecture, which itself can run on top of compressed videos. Furthermore, online video understanding works have generated fast video processing by reusing computations across frames[44,33]. These works are complementary to ours, as the fast standalone video architectures we propose can be further utilized in even more efficient online recognition.

3 | TINY VIDEO NETWORKS

In this section, we describe how to automatically design TVNs so that they can solve efficiently video understanding tasks. The goal is to build video architectures, satisfying certain constraints, e.g. runtime, by starting from random ones and iteratively improving their structure until sufficiently good architectures are obtained. We use evolution for architecture search[55,46].

To tackle video architecture search, we design the algorithm so that it can build effective combinations of layers by interleaving elements working across space and time. We specifically focus on a set of primitive neural elements, e.g. 2D spatial or 1D temporal convolutions, pooling, and other efficient layers e.g. context-gating[56], squeeze-and-excitation[57]. Video-specific elements, such as 3D convolutions, are not included as they are expensive. At the same time, these basic layers can chose to work in the spatial or temporal domain. Thus if a combination or a sequence of elements, one working across space and the other across time, is found to successful at the video understanding task, it will be selected in future evolved architectures and in the final learned architectures. The architecture search can also decide which input resolution is most advantageous, combined with how many frames per video snippet are needed and at what sampling rate. This allows for further adaptability of the framework to obtain efficient and accurate architectures. We describe the search methodology (Sec. 3.1) and the search space (Sec. 3.2).

3.1 | Tiny Video Architecture Search

In order to learn efficient video architectures, we maximize the following equation where the input is the set of parameters defining a neural network architecture. Let N_θ be the network configuration, which corresponds to a specific architecture, and θ denote the learnable weights of the network ($|\theta|$ is the number of weights in the network). Let P be a hyperparameter controlling the maximum size of the network, assuming the network will be constrained for parameter size, as well. 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 then 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. We optimize Eq. 1 by evolutionary search[55]. The search is done by measuring runtime on a regular desktop CPU. Optionally, we constrain the number of parameters or the memory footprint. The fitness function $\mathcal{F}$ we use is the sum of top 1 and top 5 accuracy per model.

We use the tournament selection evolutionary algorithm with discrete mutation operators[55]. Since the search space is large, we begin by generating a pool of 200 random networks. (Please see Section 3.2 for description of the search space). After

evaluating these networks, we randomly choose 50 of them and take the top performing network as a 'parent'. This is typically done in parallel. We then apply a discrete 'mutation' operation to this network by randomly changing one part of the network according to the search space. For example, randomly changing the input resolution, or a random layer. This allows for the 'pool' of architectures to stay diverse and at the same time to evolve to a set of potentially better architectures at each round.

The models are partially trained to 10,000 iterations in order to evaluate their performance. We ran the evolution until a saturation point of about 500 rounds. The average training time per model is about 1.5 hours, but actual training times may vary significantly, as well. Taking advantage of parallel training (we use 10 parallel workers), the full search is done within a day.

Evolutionary search provides several advantages. It allows targeting different types of devices within the search and effectively explores the irregular search space with a non-differentiable objective function, e.g. adding constraints on the parameters or memory footprint, which are important for mobile applications. Evolution allows for parallelizing the search which significantly speeds it up. Since each architecture considered in the search is extremely efficient to begin with, and networks that do not meet the timing criterion are quickly discarded, the search itself is not as computationally intensive as other architecture search methods. More extensive search, e.g., larger pool, more mutations, search over different hardware, or an expanded search space can be used. The search space can also be modified to include new elements, as we do for Mobile-specific networks in Section 6.

3.2 | Search space

In order to generate a 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 (1-32), 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 sample a sequence of layers, which are selected randomly from a potential set of components which are 2D spatial or 1D temporal convolutional layer, 1D pooling, non-local blocks[3], context-gating layers[56], and squeeze-and-excitation layers[57]. 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, e.g. the kernel sizes (from 1 to 8), strides (from 1 to 8), number of filters (from 32 to 2048) and types (e.g., standard or depthwise convolution, average or max pooling). 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). Additionally, a layer can optionally pick an activation function, a ReLu (or a swish[13] for the Mobile-friendly models). The final block is followed by a fixed standard block of global average pooling, a dropout layer (0.5 dropout rate), and a fully-connected layer which outputs the number of classes required for classification.

4 | EXPERIMENTS

We conduct experiments on four challenging datasets. TVNs are evolved on different datasets to capture various aspects of the video datasets scenarios. Models evolved on one dataset are evaluated on all other datasets, to test their usability across datasets.

Found TVN Models. By placing different constraints on the search, we automatically generate TVNs of various capacities and runtimes. Our method finds unique, yet simple and elegant architectures which are multiple times more efficient than other networks (Figure 2). Figure 3 shows some examples, where we can see the layers and their combinations selected. Interestingly, non-local layer[3] is rarely preferred in TVNs. This suggests that it is more cost-efficient to spend computation on deeper and/or wider networks. We describe the found TVNs in more details in Table 1. These models are evolved under different constraints. The models have also picked specific runtime image resolutions, number of frames f, and sampling stride s. TVN-1 is the fastest model found. It was evolved on the MiT dataset by constraining the search space to include models only running in less than 50ms on CPU. TVN-2 is evolved by limiting the search space to 100ms and 12 million parameters. TVN-3 was evolved by limiting the search space to 100ms as well, but no constraint on the number of parameters. TVN-4 is a slower model, which we created trying to understand what performance we can get allowing the model to take more time for inference. It is found by allowing networks up to 1200ms and 30 million parameters (cost roughly comparable to I3D[2]). TVN-5 is found by assigning limit on runtime and memory usage: up to 100ms and 2.5GBytes. TVN-6 is the same as TVN-5 but we set 32 frames as input (instead of 16). Thus the TVN-6 model uses the same number of frames as other models in prior art. Both models, despite using a large number of input frames, are very efficient (Figure 2), which can be attributed to also limiting the memory footprint.

TABLE 1 Description of TVN models found. Runtime is in milliseconds (ms).

Model	Evolved on	Runtime (CPU)	Runtime (GPU)	Image resol.	Num. frames (f)	Sampling rate (s)
TVN-1	MiT	37	10	224x224	2	4
TVN-2	MLB	65	13	256x256	2	7
TVN-3	Charades	85	16	160x160	8	2
TVN-4	MiT	402	19	128x128	8	4
TVN-5	MiT	86	16	160x160	16	4
TVN-6	MiT	142	18	160x160	32	4

TABLE 2 Comparison to the state-of-the-art results on Charades. We report the best TVN models in bold, and the best prior work models as bolded italics. Many TVNs, even without pretraining, outperform the SOTA which uses strong Kinetics pre-training. RGB-only results. TVNs also outperform or rival the strongest video models which use additionally flow: e.g. AssembleNet[51] 47.0 from scratch, and 53.0 with MiT pretraining; AssembleNet++[58] achieves 54.98. TVNs are also orders of magnitude faster.

Method	Runtime CPU (ms)	Runtime GPU (ms)	GFlops	mAP
Asyn-TF, VGG16[59]	-	-	-	22.4
I3D[2]	-	-	216	32.9
Nonlocal, R101[3]	-	-	544×30	37.5
SlowFast (two-stream)[27]	3594	135	213×30	42.1
SlowFast + NL (two-stream)[27]	4354	152	234×30	45.2
X3D-XL (pretr Kin-400)[52]	-	-	48.6×30	43.4
X3D-XL (pretr Kin-600)[52]	-	-	48.6×30	***47.1***
TVN-1 (from scratch)	37	10	13	40.4
TVN-2 (from scratch)	65	13	17	**47.4**
TVN-3 (from scratch)	85	16	69	**52.0**
TVN-4 (from scratch)	402	19	106	**53.8**
TVN-5 (from scratch)	86	16	52	**52.4**
TVN-6 (from scratch)	142	18	93	**52.8**
TVN-1 (MiT pretr)	37	10	13	42.1
TVN-2 (MiT pretr)	65	13	17	**48.3**
TVN-3 (MiT pretr)	85	16	69	**53.2**
TVN-4 (MiT pretr)	402	19	106	**53.9**
TVN-5 (MiT pretr)	86	16	52	**54.2**
TVN-6 (MiT pretr)	142	18	93	**54.6**

TABLE 3 Results on the MiT dataset comparing different Tiny Networks to baselines and state-of-the-art (which are all RGB-only). We report the best TVN model in bold, and the best prior work model as bolded italics. As seen, TVN models perform very competitively, but are much faster. No runtime was reported in prior works.

Method	Runtime CPU (ms)	Runtime GPU (ms)	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[60]	-	-	-	24.1%
2D ResNet-50[8]	-	-	-	27.1%
bLVNet-TAM[26]	-	-	-	***31.4***%
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%
TVN-5 (MiT)	86	16	52	29.8%
TVN-6 (MiT)	142	18	93	**30.7**%

TABLE 4 Comparison to the state-of-the-art results on MLB. No prior runtimes are available. *Our measurement of runtime.

Method	Runtime (CPU)	Runtime (GPU)	mAP
InceptionV3	-	-	47.9
I3D [11]	1865ms*	-	48.3
I3D+sub-events [11]	-	-	***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
TVN-5 (MiT)	72ms	16ms	55.3
TVN-6 (MiT)	142ms	18ms	**56.4**

TABLE 5 Performance on HMDB.

Method	GFlops	Acc. (%)
CoViAR [54]	-	59.1
I3D [2]	300	74.8
S3D-G [42]	-	***75.9***
ECO (online) [44]	64	72.4
TSM (online) [33]	65	73.5
TVN-1 (MiT)	13	72.1
TVN-2 (MLB)	17	73.5
TVN-3 (Charades)	69	71.8
TVN-4 (MiT)	106	74.7
TVN-5 (MiT)	52	73.4
TVN-6 (MiT)	93	**75.5**

4.1 | Experimental setup

Datasets. We conduct experiments on four well-established public video datasets, representing various challenges for video understanding: **Moments-in-time (MiT) [8]** is a large-scale dataset with 800k training examples and 33900 validation examples across a large number of (339) activity classes. It is a very challenging dataset, not only because of the large number of classes, but also because some of the categories are abstract or include multiple diverse concepts, e.g. dancing. **HMDB [9]** is a dataset of about 5000 training and about 1500 test examples for 51 different classes. It is a popular dataset, although relatively small (only about 100 videos per activity). **Major League Baseball (MLB) [11]** has 4290 videos for 8 different baseball activities. It is a multi-class, multi-label dataset (i.e. more than one label is correct). Unlike the previous two datasets, MLB contains longer videos and requires understanding of temporal information as the actions are fine-grained and occur in the same scene, e.g., 'bunt' and 'swing' activities are very similar. We use the segmented video setting, as in [61]. **Charades [10]** is also a multi-class multi-label dataset of about 8000 training and 1686 validation videos of 157 different in-home activities. Charades contains long, continuous videos (30 seconds on average) with multiple activities which can be occurring or co-occurring. It is also a very challenging dataset due to the large number of actions, the multi-class multi-label setting, and the small number of examples per class.

We use the established evaluation protocols for all datasets. We report runtime (on CPU and GPU), FLOPs and accuracy or mean average precision (mAP), in the context of state-of-the-art (SOTA) models. We measure runtime on an Intel Xeon CPU running at 2.9GHz and a single V100 GPU. We follow the specified network and inputs for each model: our baselines use 32 frames, as in (2+1)D ResNet [39]; I3D [2] and S3D [42] use 64 frames. We note that TVNs use fewer frames per video, e.g. 2, 8, 16 (Table 1). We report results using RGB as inputs, since flow computation itself is quite expensive and is not suitable for real-time video understanding, e.g. [32] need about 100ms on GPUs, compared to our entire TVNs running in less than 20ms.

4.2 | Performance across datasets and comparison to the state-of-the-art

Charades. Table 2 shows the performance of the Tiny Video Networks evaluated on the Charades dataset (see also Figure 2). TVNs obtain impressive results over SOTA. Firstly, TVN-2, TVN-3, TVN-4, TVN-5, TVN-6, without using any pre-training, already outperform the best state-of-the-art models which used powerful Kinetics pre-training. Among the SOTA are strong models with much more compute e.g. an architecture searched X3D [52] and the two-stream SlowFast models [27]. We also fine-tune the models trained on MiT, as is customarily done in the literature , and obtain even better results. Curiously, TVNs rival performance of the most powerful video models to date, AssembleNet and AssembleNet++ [51,58], which use additionally optical flow and connectivity learning. We note that achieving such performance at a fraction of the speed is an impressive result.

Moments-in-Time. Table 3 shows the results of TVNs on the MiT dataset (they are evolved on MiT, Charades and MLB). Since prior work did not report runtime, we also compare to strong baselines, (2+1)D ResNets [39,3] (Figure 2, right). The TVNs perform competitively or better than previous state-of-the-art methods, while running very fast. TVN-1 and TVN-2, both outperform ResNet-18 and are 57 and 33 times faster, respectively, TVN-6 outperforms ResNet-101, being 26 times faster and outperforms or performs comparatively to other state-of-the-art methods. We further note that TVN models perform very well even compared to prior *online* video models. For example, ECO and TSM [44,33] have at least three times the GFLOPs (64 and 65,

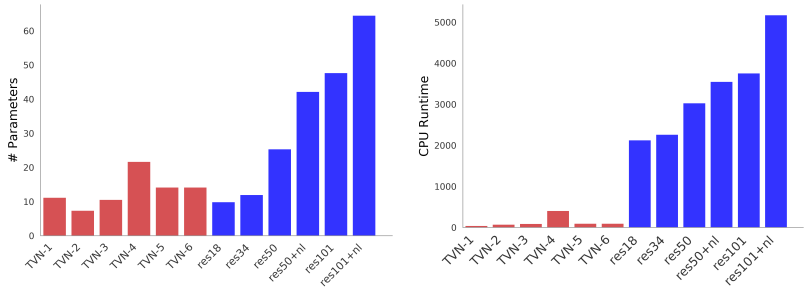

FIGURE 4 Number of model parameters (in Millions) of TVNs (in red) vs ResNets (left) and CPU runtime (right). As seen, some highly efficient nets (e.g. TVN-1 needs only 37 ms on a CPU) may actually have more parameters.

TABLE 6 TVN models obtained when expanding the search with components used in MobileNet. CPU runtime shown. MiT.

Method	Runtime	Params.	GFlops	Acc.
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%

TABLE 7 TVNs compared to MobileNets. TVNs outperform them in both faster rutimes and higher accuracies. MiT dataset.

# Frames	MobileNet Runtime	MobileNet Accuracy	TVN Runtime	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

respectively) than TVN-1 and TVN-2 (13 and 17, respectively), larger GFLOPs than TVN-5 and comparable to TVN-3. This is an opportunity to combine these two technologies for future work, and extend TVN models to online versions.

MLB. Table 4 shows the performance of TVNs on the MLB dataset, which targets fine-grained actions. We see here too that TVNs outperform SOTA. For example, TVN-5 and TVN-6 outperform I3D while being 25 and 13 times faster, respectively. TVN-2 performs comparably to I3D and is 29 times faster. TVN-6 outperforms [11], being at least 13 times faster.

HMDB. Table 5 shows the performance on HMDB. Here we report the 'averaged over 3 splits', per the standard evaluation protocol. We did not evolve on HMDB as it is a very small dataset, so all models are transferred from other datasets. TVNs are comparable to [42] on HMDB, and outperform online models ECO and TSM [44,33], where ECO further decreases its performance to 68.5 when using 16 frames, and to 61.7 when using 4 (TVN-1 achieves 72.1 with 2 frames, and TVN-4 is at 74.7 with 8).

While in general, the model evolved on its own dataset performs better than others, larger-capacity models perform best. Model runtimes do not change across datasets, as it has the same resolution and frame rate, which is sub-optimal for other datasets.

Number of parameters. Figure 4 shows the number of parameters of the main models considered in the paper. We see that, naturally, smaller models have fewer parameters. However, an interesting observation is that some of the computationally efficient TVN models have more parameters, e.g., TVN-1 has more parameters than TVN-2, but is faster. Similarly, despite TVN-3, TVN-4, TVN-5 and TVN-6 having more parameters, they are faster than counterparts ResNet-18, ResNet-34, and are also more accurate. This is due to the ability of these models to build architectures taking advantage of specific characteristics of the hardware, as the combination of layers seems to have a larger effect on runtime. TVNs were also evolved to drastically limit the parameter size or memory usage, as was done with the TVNs for Mobile applications (see Table 6).

5 | TINY VIDEO MODELS FOR MOBILE DEPLOYMENT

Mobile-friendly Tiny Video Networks. We further make a modification to our search space to include mobile-friendly components, such as inverted residual layers and the hard swish activation function, similar to MobileNet [7,13], applied both in space and time dimensions. Since search is done with the same constraints to be comparable to the original TVNs, i.e., runtime within 100ms on CPU, there is no guarantee that Mobile-only components will be selected. Still the search is able to uncover more interesting TVNs. Table 6 shows two selected mobile models, named TVN-M-1 and TVN-M-2. They are comparable to TVN-1, but achieve 23% fewer Flops and have almost twice fewer parameters, which are both important for mobile, at a small reduction in accuracy. These models satisfy all requirements for production Mobile deployment. Furthermore, we modify the original TVN-1 by substituting all ReLu activations with the hard-swish [13], we find an improvement in accuracy of 1.7% with only negligible 2ms loss in runtime, confirming its usefulness.

Comparison to MobileNet models. MobileNetV3 [13] models are among the fastest single-image models to date, and are also obtained via a neural architecture search. It is interesting to see how such models can perform when adapted to videos. We

TABLE 8 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, the original TVNs, TVN-5 performs better.

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%
TVN-5 (16 frames)	86ms	16ms	**29.8**%

TABLE 9 Increasing the number of frames on Charades. While more frames help, TVN-3 performs better.

Method	Num. frames	Accuracy
TVN-1	1 frames	39.8%
TVN-1	2 frames	40.4%
TVN-1	4 frames	41.2%
TVN-1	8 frames	42.1%
TVN-1	16 frames	43.8%
TVN-3	8 frames	**52.0**%

TABLE 10 Different methods of scaling up the model. MiT.

Method	Time CPU	Time 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%

TABLE 11 Scaling up our tiniest model (TVN-1) on MiT can reach ResNet performance.

Method	Time (CPU)	Time (GPU)	Accuracy
(2+1)D ResNet-50	3022ms	125ms	28.1%
TVN-1 (4x wide)	275ms	60ms	24.2%
TVN-1 EN	305ms	92ms	28.2%
TVN-5	72ms	16ms	29.8%
TVN-6	142ms	18ms	30.7%

compare our Tiny Video models to a MobileNetV3-equivalent ones, by applying MobileNet per frame, with a max pooling before the final fully connected layer, and training this video-adapted model. We report the performance of TVNs which have 2 and 8 frames, and of a single-frame TVN model, which we evolved on MiT, with accuracy of 20.2%, 32ms runtime, 8 GFlops, at 224x224 resolution. Table 7 shows that TVNs are advantageous in both accuracy and speed, especially notable are the significant improvements in both directions for larger number of frames.

6 | ABLATION RESULTS

Exploring a range of number of frames. One key advantage of TVNs is the opportunity to select the number of frames and how to sample them. For some datasets, MiT and HMDB, the network prefers to use very few frames 2-4 to reduce the computation cost. This is expected, given that many activities there are scene-based so a single frame is often enough. To determine the effect of temporal information, we increase the number of inputs frames used by TVN-1 from 2 to 16, and re-train these models on MiT, providing more input information to the model (Table 8). We find that this does not lead to significant performance increase, while the runtime increases a lot. At the same time our evolved TVN-5, which also has 16 frames, is 2.3 times faster and is by 6.3% more accurate on MiT. Similarly, Table 9 experiments with increasing the number of frames for Charades. Charades has more dynamic content so one can expect more frames will be beneficial. As seen, the performance grows slowly with increasing the number of frames, but falls short of the performance of TVN-3.

Scaling Up the TVNs. We further demonstrate the performance of the models by scaling up the found TVNs. In Table 10, we compare TVN-1 with increasing spatial resolution ('res'), increasing the width (number of filters in each layer) and increasing the depth (number of times each block is repeated). We scale these by multiplying them by 2 or 4. We find that some scaling lead to performance gains, but at large runtime costs. We further apply an EfficientNet-style scaling[62], i.e. scaling all dimensions (input resolution, width and depth) (Table 11). The EfficientNet-scaled model, denoted TVN-1 EN, achieves higher performance than other scaling versions, but specifically evolved TVNs, e.g. TVN-5 and TVN-6, are both more accurate and faster.

7 | CONCLUSION

We propose the Tiny Video Networks, which are automatically designed efficient video architectures. Despite working at unprecedented speeds, 10-20ms on GPU, they accomplish strong results on four challenging video datasets. Mobile-friendly versions of TVNs outperform MobileNet models adapted to video and satisfy runtime and parameter count constraints for onboard deployment. We have provided open-sourced code and models.

References

1. Tran D, Bourdev LD, Fergus R, Torresani L, Paluri M. C3D: generic features for video analysis. *CoRR, abs/1412.0767* 2014; 2(7): 8.
2. Carreira J, Zisserman A. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In: ; 2017.
3. Wang X, Girshick R, Gupta A, He K. Non-local neural networks. In: ; 2018: 7794–7803.
4. Zoph B, Le Q. Neural Architecture Search with Reinforcement Learning. In: ; 2017.
5. Pham H, Guan MY, Zoph B, Le QV, Dean J. Efficient Neural Architecture Search via Parameter Sharing. In: ; 2018.
6. Liu H, Simonyan K, Yang Y. DARTS: Differentiable Architecture Seach. In: ; 2019.
7. Sandler M, Howard A, Zhu M, Zhmoginov A, , Chen LC. Mobilenetv2: Inverted residuals and linear bottlenecks. In: ; 2018.
8. Monfort M, Andonian A, Zhou B, et al. Moments in time dataset: one million videos for event understanding. *arXiv preprint arXiv:1801.03150* 2018.
9. Kuehne H, Jhuang H, Garrote E, Poggio T, Serre T. HMDB: a large video database for human motion recognition. In: IEEE. ; 2011.
10. Sigurdsson GA, Varol G, Wang X, Farhadi A, Laptev I, Gupta A. Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding. In: ; 2016.
11. Piergiovanni A, Ryoo MS. Fine-grained Activity Recognition in Baseball Videos. In: ; 2018.
12. Piergiovanni A, Angelova A, Ryoo MS. Tiny Video Networks: Architecture Search for Efficient Video Models. In: ; 2020.
13. Howard A, Sandler M, Chu G, et al. Searching for MobileNetV3. In: ; 2019.
14. Luo JH, Wu J, Lin W. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In: ; 2017.
15. Tan M, Chen B, Pang R, Vasudevan V, Le QV. Mnasnet: Platform-aware neural architecture search for mobile. *CVPR* 2019.
16. Wofk D, Ma F, Yang TJ, Karaman S, Sze V. FastDepth: Fast Monocular Depth Estimation on Embedded Systems. In: ; 2019.
17. Wu B, Dai X, Zhang P, et al. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In: ; 2019.
18. Zhang X, Lin M, Sun J. ShuffleNet:An extremely efficient convolutional neural network for mobile devices. In: ; 2018.
19. Han Cai SH. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In: ; 2018.
20. Xiong Y, Mehta R, Singh V. Resource Constrained Neural Network Architecture Search: Will a Submodularity Assumption Help?. In: ; 2019.
21. Alwassel H, Heilbron FC, Ghanem B. Action Search: Spotting Actions in Videos and Its Application to Temporal Action Localization. In: ; 2018.
22. Carreira J, Patraucean V, Mazare L, Zisserman A. Massively parallel video networks. In: ; 2018.
23. Chen Y, Kalantidis Y, Li J, Yan S, Feng J. Multi-fiber networks for video recognition. In: ; 2018: 352–367.
24. Diba A, Fayyaz M, Sharma V, et al. Spatio-temporal channel correlation networks for action classification. In: ; 2018.
25. Diba A, Fayyaz M, Sharma V, et al. Holistic Large Scale Video Understanding. In: ; 2019.
26. Fan Q, Chen CFR, Kuehne H, Pistoia M, Cox D. More Is Less: Learning Efficient Video Representations by Big-Little Network and Depthwise Temporal Aggregation. In: ; 2019.

27. Feichtenhofer C, Fan H, Malik J, He K. SlowFast networks for video recognition. In: ; 2019.

28. Hara K, Kataoka H, Satoh Y. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?. In: ; 2018: 6546–6555.

29. Hussein N, Gavves E, Smeulders AW. Timeception for Complex Action Recognition. In: ; 2019.

30. Ji S, Xu W, Yang M, Yu K. 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2013; 35(1): 221–231.

31. Korbar B, Tran D, Torresani L. Scsampler:Sampling salient clips from video for efficient action recognition. In: ; 2019.

32. Lee M, Lee S, Son S, Park G, Kwak N. Motion feature network: Fixed motion filter for action recognition. In: ; 2018: 387–403.

33. Lin J, Gan C, Han S. TSM: Temporal Shift Module for Efficient Video Understanding. In: ; 2019.

34. Luo C, Yuille AL. Grouped spatial-temporalaggregation for efficient action recognition. In: ; 2019.

35. Qiu Z, Yao T, Mei T. Learning spatio-temporal representation with pseudo-3d residual networks. In: ; 2017: 5533–5541.

36. Simonyan K, Zisserman A. Two-stream convolutional networks for action recognition in videos. In: ; 2014: 568–576.

37. Su YC, Grauman K. Leaving some stonesunturned: dynamic feature prioritization for activity detectionin streaming video. *European Conference on Computer Vision* 2016.

38. Sun S, Kuang Z, Sheng L, Ouyang W, Zhang W. Optical flow guided feature: a fast and robust motion representation for video action recognition. In: ; 2018: 1390–1399.

39. Tran D, Wang H, Torresani L, Ray J, LeCun Y, Paluri M. A closer look at spatiotemporal convolutions for action recognition. In: ; 2018: 6450–6459.

40. Wu W, He D, Tan X, Chen S, Wen S. Scsampler:Sampling salient clips from video for efficient action recognition. In: ; 2019.

41. Wu Z, Xiong C, Ma CY, Socher R, Davis LS. Adaframe: Adaptive frame selection forfast video recognition. In: ; 2019.

42. Xie S, Sun C, Huang J, Tu Z, Murphy K. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In: ; 2018: 305–321.

43. Yeung S, Russakovsky O, Jin N, Andriluka M, Mori G, Fei-Fei L. Every moment counts: Dense detailed labeling of actions in complex videos. *International Journal of Computer Vision (IJCV)* 2015: 1–15.

44. Zolfaghari M, Singh K, Brox T. ECO: Efficient Convolutional Network for Online Video Understanding. In: ; 2018.

45. Real E, Moore S, Selle A, Saurabh Saxena YLS, Le Q, Kurakin A. Large-Scale Evolution of Image Classifiers. In: ; 2017.

46. Real E, Aggarwal A, Huang Y, Le QV. Regularized Evolution for Image Classifier Architecture Search. In: ; 2019.

47. Zhu H, An Z, Yang C, Xu K, Zhao E, Xu Y. EENA: Efficient Evolution of Neural Architecture. In: ; 2019.

48. Liu C, Zoph B, Neumann M, et al. Progressive neural architecture search. In: ; 2018.

49. Yang TJ, Howard A, Chen B, et al. NetAdapt: Platform-Aware Neural NetworkAdaptation for Mobile Applications. In: ; 2018.

50. Piergiovanni A, Angelova A, Toshev A, Ryoo MS. Evolving Space-Time Neural Architectures for Videos. In: ; 2019.

51. Ryoo MS, Piergiovanni A, , Tan M, Angelova A. AssembleNet: Searching for Multi-Stream Neural Connectivity in Video Architectures. In: ; 2020.

52. Feichtenhofer C. X3D: Expanding Architectures for Efficient Video Recognition. In: ; 2020.

53. Piergiovanni A, Ryoo MS. Representation flow for action recognition. In: ; 2019.

54. Wu CY, Zaheer M, Hu H, Manmatha R, Smola AJ, Krähenbühl P. Compressed video action recognition. In: ; 2018: 6026–6035.

55. Goldberg DE, Deb K. A comparative analysis of selection schemes used in genetic algorithms. In: Morgan Kaufmann; 1991: 69–93.

56. Miech A, Laptev I, Sivic J. Learnable pooling with Context Gating for video classification. In: ; 2017.

57. Hu J, Shen L, Albanie S, Sun G, Wu E. Squeeze-and-Excitation Networks. *CVPR* 2018.

58. Ryoo MS, Piergiovanni A, Kangaspunta J, Angelova A. AssembleNet++: Assembling Modality Representations via Attention Connections. In: ; 2020.

59. Sigurdsson GA, Divvala S, Farhadi A, Gupta A. Asynchronous Temporal Fields for Action Recognition. In: ; 2017.

60. Wang L, Xiong Y, Wang Z, et al. Temporal segment networks: Towards good practices for deep action recognition. In: Springer. ; 2016: 20–36.

61. Piergiovanni A, Fan C, Ryoo MS. Learning Latent Sub-events in Activity Videos Using Temporal Attention Filters. In: ; 2017.

62. Tan M, Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: ; 2019: 6105–6114.

How to cite this article: AJ Piergiovanni, Anelia Angelova, and Michael S. Ryoo (2021), Tiny Video Networks,