Tiny Video Networks

AJ Piergiovanni, Anelia Angelova, Michael S. Ryoo
Robotics at Google
{ajpiergi,anelia,mryoo}@google.com

Abstract

Video understanding is a challenging problem with great impact on the abilities of autonomous agents working in the real-world. Yet, solutions so far have been computationally intensive, with the fastest algorithms running for more than half a second per video snippet on powerful GPUs. We propose a novel idea on video architecture learning - Tiny Video Networks - which automatically designs highly efficient models for video understanding. The tiny video models run with competitive performance for as low as 37 milliseconds per video on a CPU and 10 milliseconds on a standard GPU.

Introduction

Video understanding is an important problem in computer vision with many applications, such as automated video tagging, activity recognition, and robot perception.

Achieving state-of-the-art results on video recognition tasks currently requires extremely large networks, often with tens to hundreds of convolutional layers, e.g., 101-layer ResNets (He et al. 2016), and looking at hundreds of frames (Wang et al. 2018). Since video tasks are quite challenging, processing both spatial (image) and temporal information, models for video understanding customarily incorporate computationally intensive modules (Tran et al. 2014; Carreira and Zisserman 2017; Xie et al. 2018; Wang et al. 2018), such as 3D convolutions, non-local blocks and others. As a result, they often suffer from very slow runtimes e.g., requiring at least 500+ ms per video snippet on a contemporary GPU and 2000+ ms on a CPU. Using such expensive networks greatly hinders their application to real-world systems, e.g., in robotics, or for mobile devices, where compute is very limited.

To address this, we propose using an evolutionary algorithm to automatically design networks that provide comparable performance at a fraction of the computational cost. More specifically, we propose a general approach which designs a family of 'tiny' neural networks for video understanding. The networks achieve competitive accuracy and run efficiently, at real-time or better speeds, within 37 to 100 ms on a CPU and 10 ms on a GPU per ∼1 second video

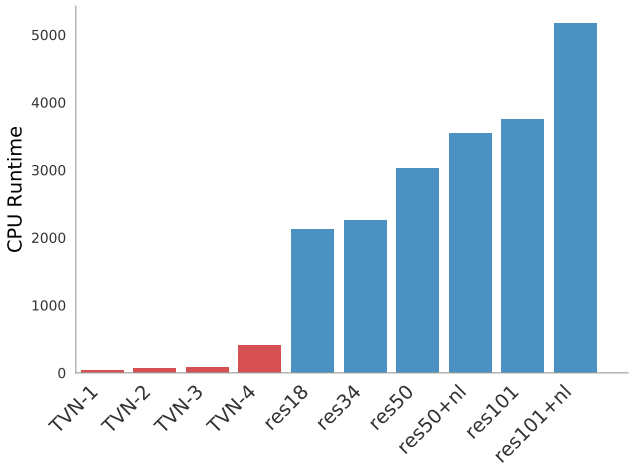

Figure 1: Tiny Video Networks (TVN) provide competitive performance at speeds as low as 37 ms on CPU/10ms on GPU per video. They are more than 100x faster than contemporary video models, such as a (2+1)D ResNet-101.

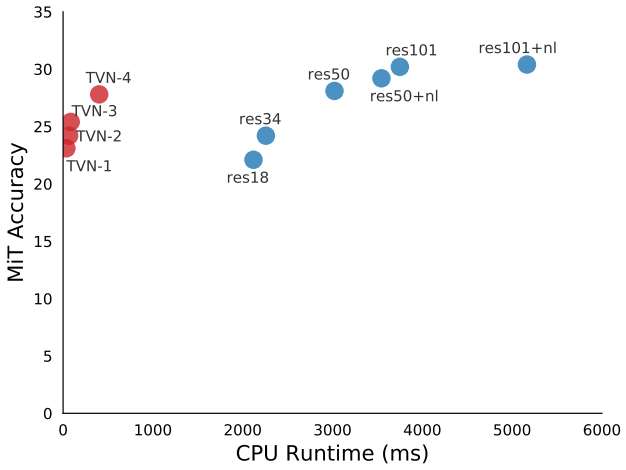

Figure 2: Runtime vs model accuracy of Tiny Video Networks compared to the main (2+1)D video understanding models on the Moments-in-Time (MiT) dataset.

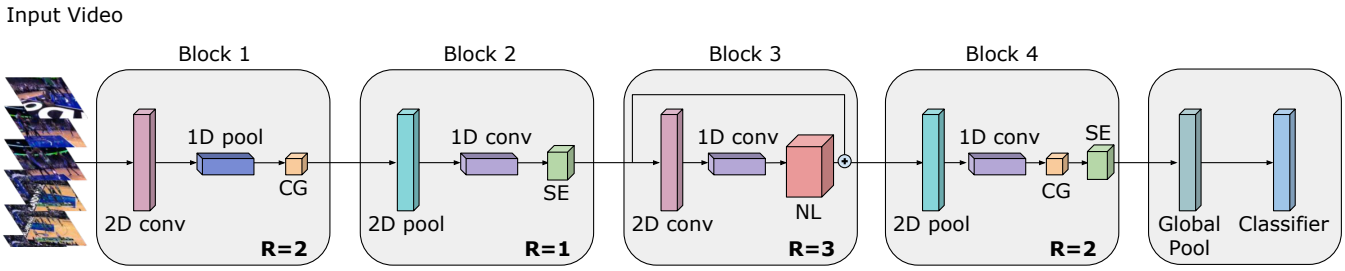

Figure 3: An example Tiny Video Network found using architecture evolution showing several blocks with different configurations. A Tiny Video Net has multiple blocks, each repeated R times. Each block has a different configuration with spatial and temporal convolution/pooling, non-local layers, context gating and squeeze-excitation layers. The final representation is globally-averaged and used as input to a fully connected layer for classification. See the supplemental material for more examples.

clip, achieving hundreds time faster speeds than contemporary models (Figures 1 and 2). We call them Tiny Video Networks (TVN), as they require extremely small runtimes, which is unprecedented for video models.

Video understanding is a very active research area with multitude of successful approaches (Ji et al. 2013; Carreira and Zisserman 2017; Simonyan and Zisserman 2014; Tran et al. 2014). Architecture search is an emerging field with many recent approaches, mostly targeting the image and language understanding domains (Zoph and Le 2017; Pham et al. 2018; Liu, Simonyan, and Yang 2019). Yet, none of these prior approaches have addressed building efficient architectures for videos. The main reasons are that videos are content-heavy and require more processing and that architecture search is a time-intensive process thus applying it to videos is non-trivial, and in many cases computationally prohibitive. In this work we address both problems, as the method produces real-time video architectures, and thus the search process itself is also efficient and does not require excessive computational resources. Furthermore, creating such fast networks opens up the opportunity both for deployment in real-life applications, e.g. online robotics systems, mobile phones, etc, as well as, for research development.

Obtaining such efficient networks is novel to video understanding, as videos contain significantly more content than images, and incorporate computationally heavy elements. This makes it very challenging to optimize video models in general. To our knowledge, no prior work has attempted finding small neural architectures for videos optimized not only for accuracy but also for runtime.

Our contributions are as follows:

- The first approach to learn highly efficient networks for videos (Tiny Video Networks). These are the fastest video networks known and run on CPU and GPU with better than real-time speeds.
- An architecture search which is designed to address the challenges of working with videos and at the same time producing fast and accurate models.
- The tiny models can further be scaled up to obtain the fastest models which work comparably to the state-of-the-art performance.

Surprisingly, the learned model architectures are different than typical video architectures with fewer convolutional layers; Tiny Video Networks prefer lightweight elements such as 2D pooling, gating layers, and squeeze-and-excitation layers (Hu et al. 2018). In contrast, contemporary video understanding models often include computationally intensive layers such as 3D convolutions. Interestingly, one of the popular non-local block modules (Wang et al. 2018) is often not preferred by the tiny networks as it gains little performance for a high computation cost. Figure 3 shows an example learned architecture.

Furthermore, our approach allows for more explorations in video architectures, at very low cost. Which will also allow for future video architecture work to be much more efficient and less computationally burdensome.

We demonstrate our results on four popular datasets for video understanding: Moments in time (Monfort et al. 2018), HMDB (Kuehne et al. 2011), Charades (Sigurdsson et al. 2016) and MLB-YouTube (Piergiovanni and Ryoo 2018). For all datasets we obtain results comparable, or slightly lower than the state-of-the-art at the fraction of the runtime. Figure 2 demonstrates that the Tiny Video Networks operate in the area of the accuracy-runtime curve where no other models exist.

Related Work

Traditionally, computationally efficient networks have been hand designed or optimized for a specific hardware (Wofk et al. 2019; Wu et al. 2019). Network architectures have also been specifically designed for mobile applications, e.g., MobileNets (Howard et al. 2017; Sandler et al. 2018; Tan et al. 2019), where larger networks are specifically optimized to be able to run at fast speeds for mobile devices.

There also have been recent attempts to automatically build time-constrained models using neural architecture search (Howard et al. 2019; Tan et al. 2019; Pham et al. 2018), sometimes taking specific hardware platforms into consideration. Some approaches also used automated fine-tuning of the network architectures with runtime in mind (Yang et al. 2018; Wu et al. 2019). Taking advantage of advances in architecture search (Zoph and Le 2017;

Real et al. 2017; Liu, Simonyan, and Yang 2019), which demonstrated large gains in recognition accuracy, prior work on searching time-constrained models obtained promising and successful results.

However, all of the above-mentioned approaches are for single-image inputs for a single-image task such as image classification or object detection. Prior study on searching for efficient and compact models for video understanding tasks such as activity recognition has been extremely limited. We focus on introducing a new framework to enable architecture search specialized for efficient video models, and present the Tiny Models found from scratch. Our proposed search space allows efficient new layer combinations to capture both spatial and temporal information in videos, different from prior work. We obtain novel efficient video architectures automatically found for the first time, which are applicable for many different video representation scenarios.

Some works have attempted to reduce the computation cost of video CNNs. Representation flow (Piergiovanni and Ryoo 2019), MFNet (Chen et al. 2018) and others reduced the computation of optical flow or motion features while CoViAR (Wu et al. 2018) focused on using compressed videos (e.g., MPEGs) to perform recognition. These works, however, still relied on heavy CNNs (e.g., ResNet-50) to obtain strong results. We focus on obtaining the efficient and compact CNN architecture, which itself runs directly on top of input videos.

Tiny Video Networks

Instead of hand-designing an architecture, we use architecture search for automatic design within a huge search space. We explore building networks constrained for both runtime and number of parameters. Further, after finding architectures satisfying those requirements, we explore how they can be scaled-up to improve performance.

Methodology

We search for the optimal combination of input resolution, both spatial (width and height) and temporal (number of frames), number of layers, their type (e.g., pooling, convolutional) and their configurations (kernel size, stride, etc). We use an evolutionary algorithm as it allows parallel evaluation and mutation of multiple individuals (i.e., networks) in the population, and effectively explores the irregular search space with a non-differentiable objective function, which here for example can be runtime. Specific search space details are in the subsection below.

In order to learn novel efficient video architecture, we maximize the following equation where the input is the set of variables/parameters defining a neural network architecture. N is the network configuration, which is defined in the below subsection. θ is the learnable parameters of the network ($|\theta|$ is the number of parameters in the network), and P is a hyperparameter controlling the maximum size of the network. $\mathcal{R}(N_\theta)$ computes the runtime of the network on a device, given the network N with its weight values θ, and R is the maximum desired computational runtime.

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

$\mathcal{F}$ is the fitness function, in our case, it measures the accuracy of the trained model on the validation set of a dataset.

We optimize Eq. 1 by evolutionary search. Note that this function is not differentiable due to the runtime and number of parameters constraints. While some works have tried to make runtime a differentiable operation, this requires estimating the runtime of each operation on the targeted device (Wu et al. 2019). However, for simplicity we chose to use an evolutionary algorithm as it allows easily targeting different devices and adding constraints on the number of parameters (e.g., model size) which is important for mobile applications. Further, once the search space is designed, it requires no tuning of other hyperparameters for the search, e.g., learning rate or loss scaling factors as in (Wu et al. 2019).

As seen in the experimental section, this algorithm discovers very efficient and at the same time accurate architectures. Also note that since each architecture considered in the search is extremely efficient to begin with, the search itself is not as computationally intensive as other neural architecture search methods.

Search Space

We designed the search space to answer the following key questions:

- What is the optimal ratio between spatial resolution (image size) and temporal resolution (number of frames)?
- How/where should the spatial and temporal information be downsampled?
- With constrained runtime, what configuration of layers provides the best performance?
- How deep/wide should the network be?

Our search space follows a simple meta-architecture consisting of multiple blocks. The number of blocks ranges between 2 and 10 and is selected as part of the search process. The final block is followed by global average pooling, a dropout layer, and a fully connected layer which outputs the number of classes required for classification. See Figure 3 for an example. In each block, the algorithm can select from a number of layers: spatial convolution, temporal convolution, non-local layers (Wang et al. 2018), context-gating layers (Xie et al. 2018), and squeeze-and-excitation layers (Hu et al. 2018).

For each layer, a number of parameters can be selected. For non-local layers, we search for the bottleneck size, as well as, representation size (e.g., size after pooling). We search for the squeeze ratio for the squeeze-and-excitation layers. 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, each block is repeated 1 to 8 times and

can have a skip/residual connection. Finally, we also include the input size as part of the search space: spatial resolution (32×32 to 320×320), number of frames to sample (1 to 128) and frame rate (1fps to 25fps).

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 extremely difficult, especially when adding constraints such as runtime or maximum number of parameters. In fact, in our experiments, we did observe that many of the architectures in this space give nearly random performance, i.e., have poor classification results, or diverge during training, or exhibit other unstable behaviors.

Evolutionary Method

We use the tournament selection evolutionary algorithm with discrete mutation operators (Goldberg and Deb 1991) as our search method.

Since the search space is large, we begin by generating 200 random networks, many of which yield poor performance. After evaluating these networks, we follow the tournament selection algorithm. From the current population of 200 networks, we randomly choose 50 of them, then 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.

Mutations. Our mutation operation simply randomly selects one part of the network and randomly changes it, as defined in the search space. This could be the input resolution, the number of blocks, or the configuration of layers within the block.

After evaluating the new network, it is added to the current population and the lowest performing network is removed. This is repeated for 1000 rounds. Each model is trained for 10,000 iterations, and since they are fast, the average training time is about 1.5 hours. When taking advantage of parallel training, this search can be done within a day.

Searching for Time Constrained Networks

Without constraining the number of parameters, we find the search generates networks with large number of parameters (e.g., >60M) which still run efficiently, by taking advantage of wide parallel layers rather than deep sequential layers. Since we are targeting mobile and robotics applications, we add an additional constraint to the maximum number of parameters in the network. This generally limits both runtime and model size.

Experiments

We conduct the following experiments:

- On 4 datasets, we compare our models to state-of-the-art results. Note that few prior video understanding work included algorithm runtimes, so we additionally included contemporary baseline methods and their runtimes.
- Placing different constraints on the search space, generating Tiny Video Networks of various capacities and runtimes.
- We explore scaling up the found Tiny Video Networks to improve performance while maintaining quick speeds.

Baselines As baselines, we compare to standard video CNNs: (2+1)D ResNets (Tran et al. 2018; Wang et al. 2018) (which are referred to simply as ResNets below), S3D (Xie et al. 2018), and I3D (Carreira and Zisserman 2017). These networks provide a good coverage of the current state-of-the-art in video recognition tasks, however most are computationally expensive and prior work does not report runtimes.

We only report results using RGB as inputs. While it is widely known that optical flow and two-stream networks is beneficial to action recognition, computing flow is quite expensive, limiting its usefulness to real-time applications. While existing works have found faster motion representations ((Piergiovanni and Ryoo 2019; Lee et al. 2018)), these layers are still expensive to apply (e.g., 100ms on GPUs) compared to our entire networks running in < 50ms on GPUs. Thus, in this work, we exclusively look at RGB-only settings.

Datasets We conduct experiments on four diverse video datasets, representing various challenges for video understanding:

- Moments-in-time (Monfort et al. 2018) is a large-scale dataset with 800k training examples and 33900 validation examples across a large number of (339) activity classes.
- HMDB (Kuehne et al. 2011) contains about 5000 training and about 1500 test examples for 51 different classes.
- MLB-YouTube (Piergiovanni and Ryoo 2018) contains 4290 videos for 8 different baseball activities. Unlike the previous two datasets where fewer or even a single frame can provide reasonable performance, MLB-YouTube requires understanding temporal information as the actions occur in the same scene and the differences are fine-grained (e.g., a 'bunt' and 'swing' activities are very similar). We use the segmented video classification setting.
- Charades (Sigurdsson et al. 2016) contains about 8000 training and 1686 validation videos of 157 different in-home activities. Unlike the other datasets, Charades contains long, continuous videos (30 seconds on average) with multiple activities which can be occurring or co-occurring.

Each of these datasets has a varying number of frames per video: Moments-in-time has about 37, MLB-YouTube has about 80 frames per video, HMDB - about 40 per video and Charades has about 350-400 frames per video. The model, as mentioned, has learned the number of frames to sample (typically between 4-8) and the frame rate which are respectively used for evaluation.

Training details We train our models for 50,000 steps using a batch size of 128. We use a cosine decay for the learning rate with a maximum value of 2.2 and a linear warmup for 4000 steps. We apply dropout with a probability of 0.5 before the classification layer. We use weight decay scaled by 1×10^{-7}. For Moments-in-time and HMDB, we use a softmax activation function and cross-entropy loss. For Charades and MLB-YouTube, we use the sigmoid activation function as they are multi-label datasets.

Found TVN Models

We compare four of 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 a slower model, found by limiting the search space to 100ms and 12 million parameters (it runs at 65ms and was evolved on MLB-YouTube). 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 was 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 are evolved on different datasets to capture various aspects of the specific video dataset scenarios. In our experiments below, we report their performances on all or most datasets, despite them being evolved on specific ones, so as to test their usability across datasets.

Tiny Video Networks - Main Results

Tables 1, 2, 3, and 4 show the performance of the Tiny Video Networks evaluated on the four datasets for video understanding. We report runtime (both on CPU and GPU) and accuracy, in the context of state-of-the-art 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; i.e., (2+1)D ResNet (Tran et al. 2018) use 32 frames, I3D (Carreira and Zisserman 2017) and S3D (Xie et al. 2018) use 64 frames.

Table 1 shows the main results on the Moments-in-Time dataset. We compare the Tiny Video Model results against baselines for various datasets. Our significantly faster TVNs perform similarly to previous state-of-the-art methods at a fraction of the cost. We note that achieving competitive or better performance at a fraction of the speed is an impressive result on its own. For example, Tiny Video Networks, with 23.1 and 24.2 accuracy, both outperform ResNet-18 at 57 and 33 times the speed and are at the same performance as ResNet-34, while being 61 and 34 times faster, respectively. TVN-1 is 100 times faster than the commonly used ResNet-101 model for videos.

In Table 2 we show result for the MLB-YouTube dataset, similarly reporting runtime, accuracy, and comparing to the state of the art. We observe here too that we can obtain very competitive results 44-52 percent accuracy for models working within 19 ms on a GPU. For this dataset, as well as the other two, Charades and HMDB, we fine-tune the models trained on Moments-in-time, as it customarily done in the literature. Previous work as reported in the table has pretrained on a Kinetics dataset, which is no longer available in its complete form, so we use MiT as alternative.

Table 1: Results on the Moments-in-time dataset comparing four different Tiny Networks to baseline (2+1)D ResNets and state-of-the-art results from †(Monfort et al. 2018). The Tiny Networks achieve similar performance at a fraction of the compute cost. No runtime was reported in the literature.

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
ResNet-18	2120ms	105ms	21.1%
ResNet-34	2256ms	110ms	24.2%
ResNet-50	3022ms	125ms	28.1%
ResNet-101	3750ms	140ms	30.2%
2D ResNet-50 †	-	-	27.1%
Two-stream I3D †	-	-	29.5%
TVN-1	37ms	10ms	23.1%
TVN-2	65ms	13ms	24.2%
TVN-3	85ms	16ms	25.4%
TVN-4	402ms	19ms	27.8%

Table 2: Comparison to the state-of-the-art results on MLB-YouTube (RGB-only). Prior results are obtained from (Piergiovanni and Ryoo 2018) (shown in the upper part) and for I3D we measured the runtime ourselves (denoted as *).

Method	Runtime (CPU)	Runtime (GPU)	mAP
InceptionV3	-	-	47.9
I3D	1865ms*	-	48.3
I3D+sub-events	-	-	55.5
TVN-1	37ms	10ms	44.2
TVN-2	65ms	13ms	48.2
TVN-4	402ms	19ms	52.3

In Table 3, we compare our models on the Charades dataset (Sigurdsson et al. 2016), and Table 4 presents the results on HMDB. Here two we observe that the models perform similarly well, despite being very efficient. This confirms for us that that these models are universal, applicable across many scenarios.

Runtimes and number of parameters

Figure 1 visualizes the runtimes of TVN in the context of prior methods. We can clearly see that the TVN models are significantly faster than all others. Figure 2 further shows the runtime vs the accuracy of TVNs, together with prior methods. As seen here, while not the most accurate in absolute terms, they are more accurate than models much slower than them and provide very good accuracy-speed trade-off.

We further note that the runtime of the Tiny Video Networks are very impressive for videos. To put this in context, without attempting a direct comparison because image sizes and frames vary, it is comparable to single-image-only mod-

Table 3: Comparison to the state of the art results on Charades. Prior work shown in that order in the upper part: (Wu et al. 2018; Sigurdsson et al. 2017; Wang et al. 2018); they do not report runtime. I3D is our baseline. *Our measurement of runtime.

Method	Runtime (CPU)	Runtime (GPU)	mAP
CoViAR, Res-50	-	-	21.9
Asyn-TF, VGG16	-	-	22.4
I3D	-	-	32.9
Nonlocal, R101	4850ms*	157ms*	37.5
TVN-1	37ms	10ms	32.2
TVN-3	85ms	16ms	33.5
TVN-4	402ms	19ms	35.4

Table 4: Comparison to the state of the art results on HMDB (RGB-only). Most prior work is shown in the upper part and does not report runtime: (Wu et al. 2018; Carreira and Zisserman 2017; Xie et al. 2018).

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
CoViAR, Res-50	-	-	59.1
I3D	-	-	74.8
S3D-G	-	-	75.9
TVN-1	37ms	10ms	72.1
TVN-2	65ms	13ms	73.5
TVN-4	402ms	19ms	74.7

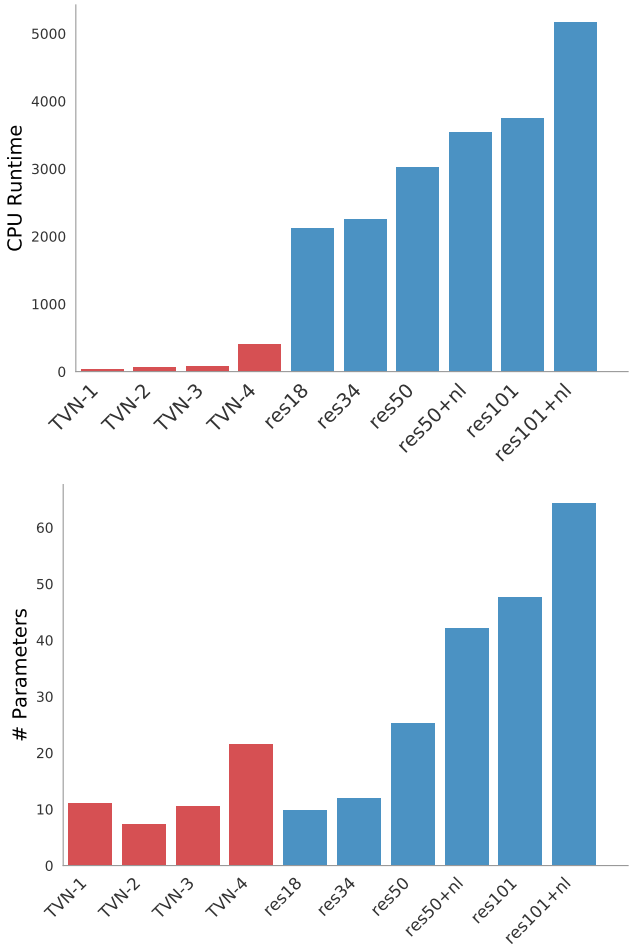

Figure 4: Runtime (top) and number of model parameters (bottom) of Tiny Video Nets vs ResNet vs others. As seen, a highly efficient Tiny Video Networks (only 37 ms on a CPU) may actually have more parameters, which is unexpected, but was automatically designed by our algorithm to obtain highly optimized performance.

els developed for mobile applications. For example, the runtime of MobileNet V2 (Sandler et al. 2018) ranges between 21ms and 37ms for a single frame input.

Figure 4 shows both the runtime and the number of parameters of corresponding models. We see that naturally smaller models do have fewer parameters. However, an interesting observation is that some of the computationally efficient models have larger parameter sizes, e.g. TVN-1. Also another interesting observation is that despite TVN-3 and TVN-4 having more parameters, they are significantly faster than counterparts ResNet-18 and ResNet-34, and also are more accurate. TVN models tend to have fewer parameters in general than other contemporary models, in addition to being much faster (Figure 4, top sub-figure).

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 well.

To determine the effect of temporal information on performance, we increased the number of frames used by TVN-2 from 4 to 8 and 16. We further conducted a search requiring every network to have 16 frames as input. This was designed to find the best way to use temporal information for fast CNNs.

Our results are shown in Table 5. We find that increasing the number of frames for TVN-2 on Moments in time does not lead to significant performance increase, while the runtime does increase. When searching with 16 frames, we find the model prefers to use temporal pooling early to reduce the computation cost. However, this model does not perform as well as TVN-2.

Findings

Our main finding is that Tiny Video Networks have fast runtimes on both CPUs and GPUs with very good accuracy. Furthermore, our method is capable of generating multiple models on the range of efficiency-performance spectrum.

Table 5: Increasing the number of frames for TVN-2 from 4 to 16 compared to a search forcing all models to use 16 frames. On Moments in Time, we find that more frames is not greatly beneficial.

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
TVN-2 (4 frames)	65ms	13ms	23.1%
TVN-2 (8 frames)	140ms	28ms	23.4%
TVN-2 (16 frames)	200ms	45ms	23.5%
TVN-16 frames	67ms	14ms	22.2%

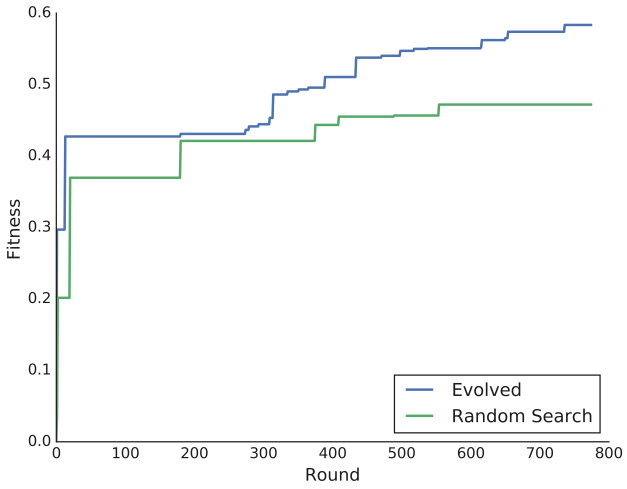

Figure 5: Random search vs. evolution for our tiny video net search space. We find that evolution yields better models more quickly.

Figure 3, visualizes an example model, showing TVN-1, see the supplemental results for examples for the other generated Tiny Video Networks. We also find that evolution is beneficial compared to random search (see Fig. 5), due to our large search space which contains many bad networks.

One interesting finding is that the popular non-local layer (Wang et al. 2018) is never preferred in Tiny Video Architectures. This suggests that it is more cost-efficient to spend computation on deeper and/or wider networks with larger inputs. We also find that layers such as context gating are commonly used, which are significantly cheaper than the non-local layer, but still capture some similar global-attention features.

Scaling Up the TVNs

We further demonstrate the performance of the models by scaling up the found Tiny Video Networks. In Table 6, 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 the most performance gains.

Table 6: Different methods of scaling up the model on Moments in Time. We explore scaling up spatial resolution (res), width, and 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%

Table 7: Scaling up our TVN-1 model on MiT based on EfficientNet coefficients (denoted as TVN-1 EN), and compared to state-of-the-art models. * runtime measured on our machine. † numbers from (Monfort et al. 2018).

Method	Runtime (CPU)	Runtime (GPU)	Accuracy
(2+1)D ResNet-50	3022ms	125ms	28.1%
(2+1)D ResNet-101	3750ms	140ms	30.2%
2D ResNet-50 †	1034ms*	53ms*	27.1%
Two-stream I3D †	-	-	29.5%
TVN-1	65ms	13ms	23.1%
TVN-1 EN	305ms	92ms	28.2%

Based on the findings of EfficientNet (Tan and Le 2019), we scaled up TVN-1 in all dimensions (input resolution, width and depth) based on their coefficients. In Table 7, we compare our scaled up network to state-of-the-art results. Our scaled up model, denoted as TVN-1 EN, is able to achieve comparable performance to much larger models, still being very efficient.

Conclusion and future work

We present a novel approach for automatically learning Tiny Video Network architectures. These are the first video models that we are aware of which are both fast and accurate. They are automatically discovered, perform well as seen on all four datasets, and provide a good trade-off in speed vs. accuracy. The models are hundreds of times faster than contemporary video models and have fewer parameters. As such they can be applied to real-time robotics applications and run on mobile devices.

One interesting observation of the paper is that Tiny Video Networks, when scaled up, perform very close or on par with the best state-of-the-art models, while being multiple times faster. It is possible to further explore these models, and obtain even better accuracy at the fraction of the speed.

References

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

Chen, Y.; Kalantidis, Y.; Li, J.; Yan, S.; and Feng, J. 2018. Multi-fiber networks for video recognition. In *Proceedings of European Conference on Computer Vision (ECCV)*, 352–367.

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

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; and Marco Andreetto, H. A. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *CoRR:1704.04861*.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Ruoming Pang, V. V.; Le, Q. V.; and Adam, H. 2019. Searching for mobilenetv3. In *CoRR:1905.02244*.

Hu, J.; Shen, L.; Albanie, S.; Sun, G.; and Wu, E. 2018. Squeeze-and-excitation networks. *CVPR*.

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

Kuehne, H.; Jhuang, H.; Garrote, E.; Poggio, T.; and Serre, T. 2011. Hmdb: a large video database for human motion recognition. In *ICCV*. IEEE.

Lee, M.; Lee, S.; Son, S.; Park, G.; and Kwak, N. 2018. Motion feature network: Fixed motion filter for action recognition. In *Proceedings of European Conference on Computer Vision (ECCV)*, 387–403.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable architecture seach. In *ICLR*.

Monfort, M.; Andonian, A.; Zhou, B.; Ramakrishnan, K.; Bargal, S. A.; Yan, T.; Brown, L.; Fan, Q.; Gutfruend, D.; Vondrick, C.; et al. 2018. Moments in time dataset: one million videos for event understanding. *arXiv preprint arXiv:1801.03150*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

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

Piergiovanni, A., and Ryoo, M. S. 2019. Representation flow for action recognition. In *CVPR*.

Real, E.; Moore, S.; Selle, A.; Saurabh Saxena, Y. L. S.; Le, Q.; and Kurakin, A. 2017. Large-scale evolution of image classifiers. In *ICML*.

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

Sigurdsson, G. A.; Varol, G.; Wang, X.; Farhadi, A.; Laptev, I.; and Gupta, A. 2016. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *Proceedings of European Conference on Computer Vision (ECCV)*.

Sigurdsson, G. A.; Divvala, S.; Farhadi, A.; and Gupta, A. 2017. Asynchronous temporal fields for action recognition. In *CVPR*.

Simonyan, K., and Zisserman, A. 2014. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems (NeurIPS)*, 568–576.

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

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; and Le, Q. V. 2019. Mnasnet: Platform-aware neural architecture search for mobile. *CVPR*.

Tran, D.; Bourdev, L. D.; Fergus, R.; Torresani, L.; and Paluri, M. 2014. C3d: generic features for video analysis. *CoRR, abs/1412.0767* 2(7):8.

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

Wang, X.; Girshick, R.; Gupta, A.; and He, K. 2018. Non-local neural networks. In *CVPR*, 7794–7803.

Wofk, D.; Ma, F.; Yang, T.-J.; Karaman, S.; and Sze, V. 2019. Fastdepth: Fast monocular depth estimation on embedded systems.

Wu, C.-Y.; Zaheer, M.; Hu, H.; Manmatha, R.; Smola, A. J.; and Krähenbühl, P. 2018. Compressed video action recognition. In *CVPR*, 6026–6035.

Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; and Keutzer, K. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*.

Xie, S.; Sun, C.; Huang, J.; Tu, Z.; and Murphy, K. 2018. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of European Conference on Computer Vision (ECCV)*, 305–321.

Yang, T.-J.; Howard, A.; Chen, B.; Zhang, X.; Go, A.; Sandlerand, M.; Sze, V.; and Adam, H. 2018. Netadapt: Platform-aware neural networkadaptation for mobile applications. In *Proceedings of European Conference on Computer Vision (ECCV)*.

Zoph, B., and Le, Q. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

Found TVNs

In Figs. 6, 7, 8 and 9, we show examples of the 4 found TVN models. Each found different structure that captures different features while still running quickly.

Different temporal scales. We also found that searching on different datasets produces models with different temporal scales. For example, when searching on Moments-in-Time, which has short 3 second videos, the model prefers to use 2 or 4 frames at covering about 1 second of video. However, when searching on Charades, which has long 30-second videos, the model generally selects 8 frames covering about 10 seconds of video or 24 frames covering 30 seconds of video.

Similarly, on MLB-YouTube, the models cover about 6 to 8 frames of video, but only capture about 2 to 5 seconds, reflecting the short duration of the videos but indicating that temporal information is more important on this dataset than Moments-in-Time.

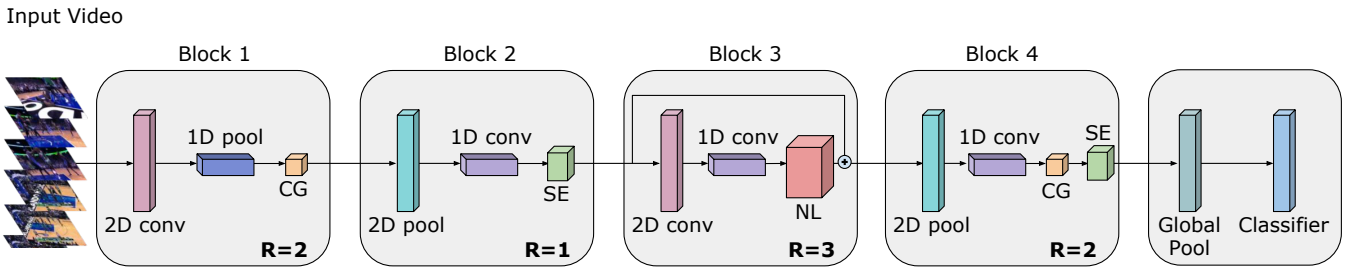

Figure 6: Same as in main paper. An illustration of TVN-1 found using architecture evolution showing several blocks with different configurations. A Tiny Video Net has multiple blocks, each repeated R times. Each block has a different configuration with spatial and temporal convolution/pooling, non-local layers, context gating and squeeze-excitation layers. The final representation is globally-averaged and used as input to a fully connected layer for classification.

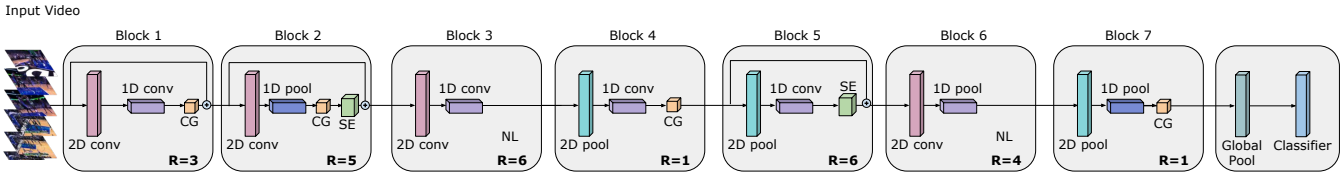

Figure 7: Illustration of TVN-2.

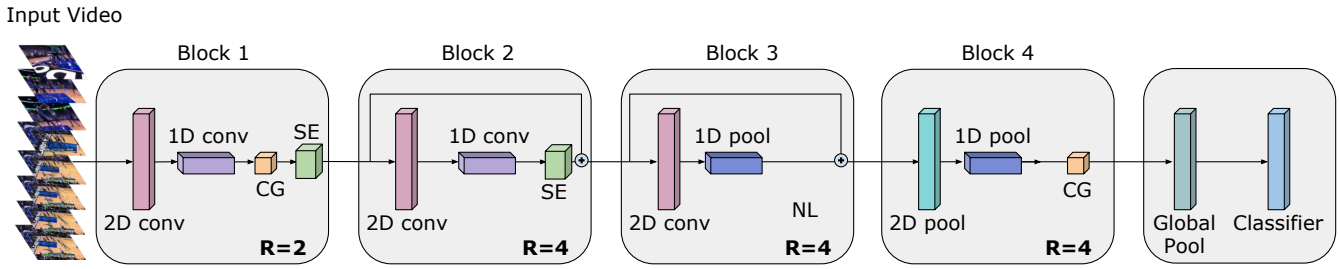

Figure 8: Illustration of TVN-3.

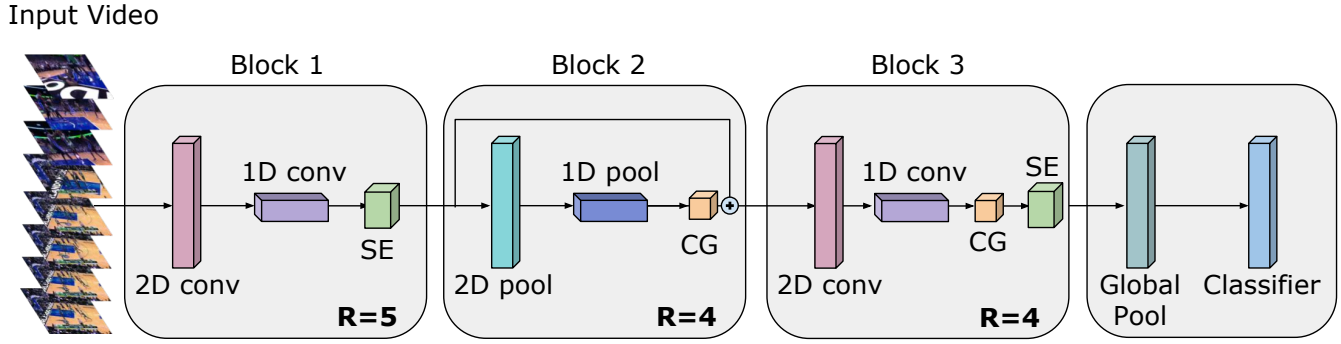

Figure 9: Illustration of TVN-4.