A practitioner's guide to improve the logistics of spatiotemporal deep neural networks for animal behavior analysis

Lakshmi Narasimhan Govindarajan Rohit Kakodkar Thomas Serre
{lakshmi_govindarajan, rohit_kakodkar, thomas_serre}@brown.edu
Brown University, Providence, RI, USA

***Abstract*—Quantifying behavior is essential for understanding how the brain works. Advances in high-throughput video monitoring capabilities combined with the development of sophisticated spatiotemporal deep neural networks offer tremendous promise for high-throughput behavioral neuroscience research. However, the high cost associated with the deployment of these systems has restricted their widespread applicability. Here, we explore a suite of optimization techniques for a representative neural network architecture, the I3D, trained to perform action classification on freely behaving mice in a homecage setup. Our results suggest that simple optimizations in data loading protocols and network specification yield significant reductions (up to $\sim 50\%$) in model runtime (and hence in associated computing costs) without sacrificing the system's overall accuracy.**

I. Introduction

Behavior reflects a cascade of neuronal processes and constitutes the ultimate brain output. Quantifying behavior in naturalistic contexts is thus vital to mainstream neuroscience research and it is also critical for biomedical research including drug discovery and disease monitoring. One of the primary challenges in quantifying behavior is phenotypic diversity. Each individual's behavioral dynamics span from timescales of milliseconds to hours with a high degree of variability [3, 12]. Capturing this diversity necessitates continuous, high-throughput, and high-resolution video imaging made possible by recent engineering advances [2]. Naturally, the vast trove of video data provided a thrust for the development of algorithms for automating behavioral analysis [10, 17, 1, 6, 18, 14, 11].

Until recently, the vast majority of computer vision research has heavily focused on the analysis of images including image categorization and segmentation, object detection and face recognition among others [21, 8, 20]. Models that approached human accuracy on such tasks are derivatives of a general class of feedforward artificial neural architectures known as Deep Convolutional Networks (DCNs). As a result of the enormous success of DCNs in computer vision, they were quickly adopted in behavioral research to automate many of the image analysis pipelines from animal detection, counting, and pose estimation [15, 14].

Dealing directly with continuous video data, however, brings a new challenge because of the enormous amount of data that needs to be processed for video data compared to image data [22]. A natural extension of DCNs from 2D spatial to 3D spatiotemporal neural architectures resulted in orders of magnitude increase in the parametric complexity of these algorithms. As a result, neural networks for video analysis are significantly slower, need more data to train, and exhibit poorer generalization capabilities compared to their 2D spatial counterparts. In practice, this has limited the applicability of these architectures for domains such as behavioral neuroscience with enormous amounts of data to be analyzed. As a result, current methods extract frame-based embeddings and use adhoc strategies to integrate information from these frame-level features over time [19, 24]. More recently, the rise of large-scale, naturalistic video datasets has started to afford the learning of high-quality spatiotemporal representations that serve as good starting points for several downstream objectives.

The runtime complexity (even for inference) of deep spatiotemporal models, however, remains prohibitive to their utility for automated continuous behavior monitoring. We consider the case where a typical behavioral experiment comprises a cohort of N animals, each recorded continuously for T days, with *one* such experiment resulting in $N \times T \times 24$ hours of video. For a reasonable choice of $N = 24$ and $T = 5$ (as done in [9, 23, 6]), we are faced with the processing of $2,880$ hours of video. A back-of-the-envelope calculation reveals this to be $\sim 1,100$ GPU hours on a standard Nvidia Tesla V100 (or nearly 46 days of compute / wait time). Practicality demands that inference on this collection of videos is executed in parallel, which necessitates the need for a very significant computing infrastructure. Though most research laboratories lack local infrastructure at this scale, they can leverage cloud computing platforms such as Google Cloud Platform (GCP) and Amazon Web Services (AWS). Even at a nominal rate of $\$1.50$ per hour, the aforementioned behavioral experiment will result in a total cost of $\sim \$1,650$! This exercise highlights the need for techniques to trim model runtimes towards reasonable wait times and associated costs.

In the remainder of this short article, we consider a standard spatiotemporal DCN, the I3D [4], that was trained to perform a 9-way behavior classification on videos of freely behaving mice in homecages [7]. The overall balanced accuracy of the system was at 81.5% (with chance performance being 11.1%), reaching the inter-human agreement ceiling and comparable to successful studies in the past [10, 9, 23, 6]. Herein, we focus on summarizing the general insights we have gained from our attempts to improve the runtime efficiency of this model, with an eye on large-scale inference. Specifically, we were able to take advantage of computations that can be executed asynchronously in parallel as well as acceleration through the TensorRT framework. Leveraging these optimizations, we were able to reduce the model runtime and costs by up to $\sim$ 50% while preserving the system's overall accuracy. While this work focuses on the I3D architecture and our specific behavioral analysis workflow, the approach is general. We make available code snippets to highlight the ease of incorporating these optimizations into any workflow.

Contributions: We consider the application of a standard spatiotemporal DCN, the I3D, for the automated continuous behavior monitoring of mice. (1) We explore a set of simple modifications to significantly improve the model runtime without compromising accuracy; (2) we demonstrate the associated benefits in terms of reducing the financial cost of running experiments; and (3) we make available code snippets which can be easily added to standard code bases for automated behavioral analysis.

II. The synchronous pipeline

The most basic workflow adopted during the inference phase of our I3D, or any other spatiotemporal DCN for that matter, comprises three steps: (1) Reading a "minibatch" of inputs ($\mathbf{b_i}$) into GPU memory either from files on disk or from onboard CPU memory; (2) applying the necessary pre-processing steps on this tensor; and (3) executing a forward pass through the I3D with the pre-processed minibatch. In our specific use case, $\mathbf{b_i} \in \mathrm{R}^{64\times3\times480\times640}$, where the leading dimension denotes the temporal size of an input chunk. The pre-processing operations include resizing the input tensor to canonical dimensions, and normalizing the input values. The network forward passes utilize GPU compute, while data loading and pre-processing are predominantly CPU-bound. Without custom specification however these operations are done synchronously thus resulting in increased GPU idle time (Fig. 1a). This situation is sometimes referred to as being I/O-bound meaning that the GPU is "waiting" for data to arrive. Similarly, the CPU cores remain idle when the network forward pass is being executed. We note that cost-cutting changes here, even minimal, scales with the inference dataset size.

Towards addressing this, we start by rewriting our naive data loader method to instead inherit and wrap over the Tensorflow `Dataset` class [13][1]. The `Dataset` API supports iterating through data in a streaming fashion while applying necessary transformations in addition to other useful functionalities that we will shortly expose.

To provide a handle on converting one's naive data loader to utilize the `Dataset` API, we provide code snippets for both a basic data loader routine (Snippet 1) and the equivalent `Dataset` implementation (Fig. 1 (Bottom)). We observe that the changes required are minimal and user-friendly.

[1]https://www.tensorflow.org/api_docs/python/tf/data/Dataset

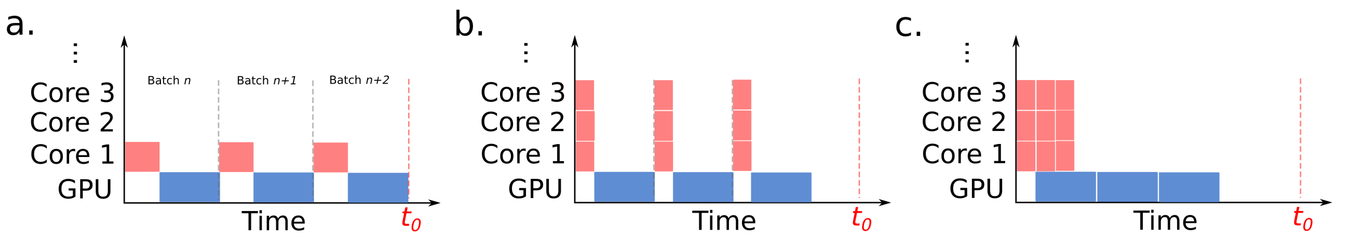
Snippet 1 A naive data loader for spatiotemporal inputs.

```
def VideoIterator(video_file, batch_size):
  video = open(video_file)
  batch, idx = [], 1
  for frame in video:
    pre-processed_frame = pre-process(frame)
    batch.append(pre-processed_frame)
    if (idx == batch_size):
      yield batch
      batch = []
      idx = 0
    else:
      idx += 1
```

III. Parallelism and asynchronous fetching

As we note above, the `Dataset` API supports a host of useful functionality to cut down on idle times. One simple advancement is to distribute the load of fetching and pre-processing a batch of inputs over multiple CPU cores (Fig. 1b). This is carried out via the `map` protocol as highlighted in Fig. 1b (Bottom). Though this offers marginal performance benefits, it does not address the GPU-idling bottleneck. Asynchronous data loaders offer a solution to this particular problem. In essence, the idea of pre-fetching batches entails the preparation of future minibatches of input with CPU cores while the current minibatch is being processed by the GPU (Fig. 1c). In the most effective of cases, when a GPU is done processing a minibatch, the next minibatch is ready for use. This cuts down the GPU-idle time and puts the model in a regime where it is compute-bound. The `pre-fetch` routine in the `Dataset` API implements this functionality.

We also analyse the performance gains of the parallel and asynchronous pipeline as a function of the number of CPU cores allotted (Fig. 2). Our asynchronous system eventually saturates with increasing number of CPUs indicating that at some point the inter-thread communication overhead outweighs the benefits of wider distribution. At our optimal configuration, the asynchronous pipeline achieves a $\sim$ 50% reduction in runtime as compared to the baseline synchronous processing. We note that all the improvements specified thus far purely pertain to increasing CPU efficiency and reducing GPU idle time. To contextualize the scope of further *model* optimization, we compute a lower-bound measure for runtime (Fig. 2). This measure indicates the amount of reduction in runtime possible by optimizing the I3D architecture directly, and thereby increasing GPU efficiency. Though a deep dive into techniques such as model distillation [16] and pruning [5] is beyond the scope of this paper, we provide a glimpse into benefits that can be had by adjustments to the network's floating point precision.



```
class VideoIterator(tf.data.Dataset)
    def _generator(video_file):
        video = open(video_file)
        for frame in video:
            preprocessed_frame = preprocess(frame)
            yield preprocessed_frame

    def __new__(cls, video_file, batch_size):
        return tf.data.Dataset.from_generator(
            cls._generator,
            args = ([[video_names]])) \
            .batch(batch_size)
```

```
class VideoIterator(tf.data.Dataset)
    def _generator(video_file):
        video = open(video_file)
        for frame in video:
            yield frame

    def __new__(cls, video_file, batch_size):
        return tf.data.Dataset.from_generator(
            cls._generator,
            args = ([[video_names]])) \
            .map(preprocess,
                num_parallel_calls =
                tf.contrib.data.AUTOTUNE) \
            .batch(batch_size)
```

```
class VideoIterator(tf.data.Dataset):
    def _generator(video_file):
        video = open(video_file)
        for frame in video:
            yield frame

    def __new__(cls, video_file, batch_size):
        return tf.data.Dataset.from_generator(
            cls._generator,
            args = ([[video_names]])) \
            .map(preprocess,
                num_parallel_calls =
                tf.contrib.data.AUTOTUNE) \
            .batch(batch_size) \
            .prefetch(tf.contrib.data.AUTOTUNE)
```

Fig. 1. Profiling the distribution of compute time across CPU cores and a GPU. Data loading and tensor pre-processing are typically CPU-based operations while network forward passes utilize GPU compute. (Top) (a) With a synchronous data-loader the CPU utilization and GPU utilization are interleaved resulting in excess GPU idle time. (b) Multiprocessing can decrease CPU wall times but cannot prevent the GPU from idling. (c) Asynchronous data loading alleviates this issue. (Bottom) Code snippets demonstrating how each of these functionalities can be specified using the Tensorflow `Dataset` API. Changes are highlighted in red.

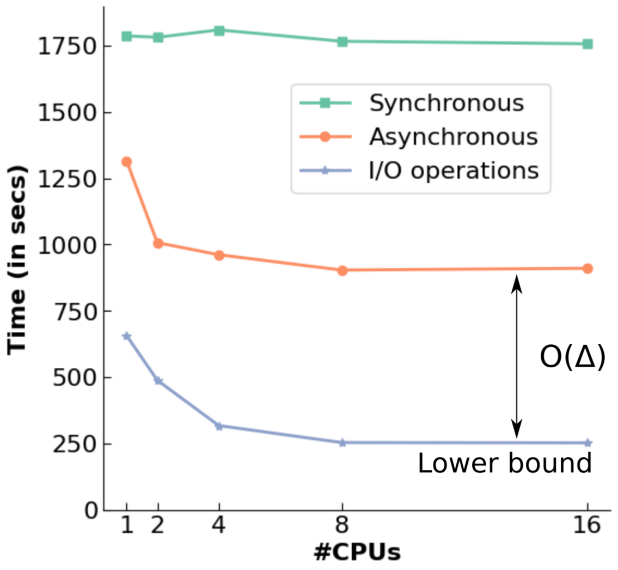


Fig. 2. The diminishing utility of invoking multiple CPU cores. On the one hand, the synchronous data processing pipeline is, as expected, agnostic to the number of CPU cores invoked. On the other hand, the asynchronous pipeline shows performance gains with increasing number of CPU cores before eventually saturating. We also compute a lower-bound for runtime by just calculating the time to asynchronously load data without any neural network inference. All timings here correspond to processing a single 1-hr long video.

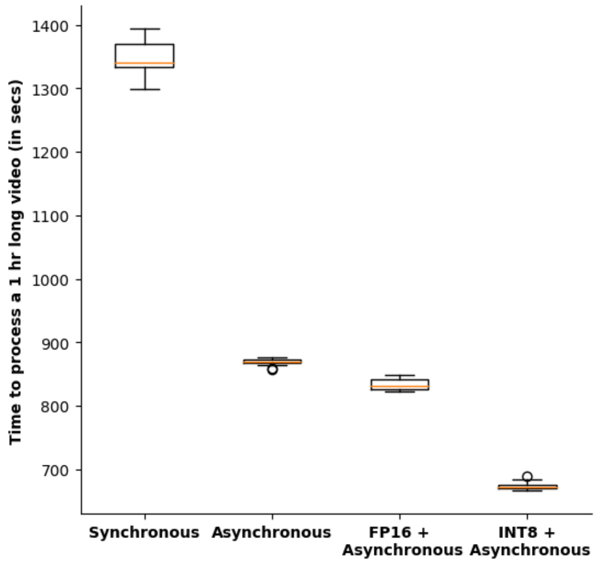


Fig. 3. Profiling the runtime efficiency of our I3D model with floating point adjustments on top of the asynchronous data loader. Runtime data was collected from $N = 30$ one hour long videos processed on a standard Nvidia Tesla V100 GPU with a batch size of 4. FP16 and INT8 refer to quantizations (both parameters and network activations) that use reduced precision of 16-bits and 8-bits respectively.

IV. Acceleration through TensorRT

TensorRT[2] is an Nvidia library specifically designed for inference acceleration. Among several standout features offered by TensorRT, herein, we focus on the benefits of reduced precision and quantization for inference runtime. Most major neural network libraries treat model parameters and activations as 32-bit floating point numbers by default. Though this high degree of precision is beneficial for optimization and network stability during *training*, it may be superfluous for inference. Reduced precision inference can significantly improve throughput due

[2]https://developer.nvidia.com/tensorrt

TABLE I
PRICING ESTIMATES (PER HOUR, PER VIDEO) ON GCP FOR RUNNING INFERENCE IN EACH MODEL CONFIGURATION. THE OVERALL REDUCTION IN SYSTEM ACCURACY FOR THE OPTIMIZED VERSION WAS WITHIN 2% OF THE ORIGINAL.

Configuration	Cost per hour per video
Synchronous	\$0.93
Asynchronous	\$0.60
FP16 + Asynchronous	\$0.57
INT8 + Asynchronous	\$0.46

to its smaller computational demands. However, simply truncating floating-point values can be calamitous.

TensorRT specializes in taking a trained neural network model and constructing an optimized inference engine while calibrating for reduced precision. Here we tested the runtime performance of I3Ds adjusted to operate on 16-bit floats (FP16) and 8-bit integers (INT8) through TensorRT. Though we do not make it explicit, we oberve and verify that all adjustments yield approximately the same level of task accuracy. We report runtimes on a standard Nvidia Tesla V100 GPU in Fig. 3.

In Table I, we have translated runtimes into pricing estimates as listed for Nvidia Tesla V100 GPUs on the Google Cloud Computing platform. Given the vast of amount of computing associated with continuous monitoring, even marginal computational savings on a single network's forward pass yields significant cost reduction. As a case in point, let us revisit our hypothetical behavioral experiment from before. Performing deep neural inference on *just* this dataset amounts to a difference of $\sim$ \$1300 between the Synchronous and INT8+Asynchronous pipelines!

V. Discussion and Conclusion

In this work, we explored the practical cost of using a state-of-the-art spatiotemporal deep neural network model, the I3D, for automated continuous mice behavior analysis. Despite the promise of this approach, widespread adoption by the behavioral neuroscience community is lacking because of the high cost due to the computational needs of these architectures. To this end, we explore a suite of optimization tricks towards improving the runtime efficiency, and associated deployment costs of these systems. In particular, we find that optimizing data loading routines via parallel and asynchronous data loading techniques combined with reduced precision inference facilitated by the TensorRT framework yield reduction in costs by nearly 50%. As we point out earlier, model compression and knowledge distillation techniques can further improve the cost-effectiveness of spatiotemporal models for large-scale inference. These are active research topics themselves, but do hold promise for behavioral neuroscience in the future.

References

[1] Ahmet Arac et al. "DeepBehavior: A deep learning toolbox for automated analysis of animal and human behavior imaging data". In: *Frontiers in systems neuroscience* 13 (2019), p. 20.

[2] Ida L Barlow et al. "Megapixel camera arrays enable high-resolution animal tracking in multiwell plates". In: *Communications biology* 5.1 (2022), pp. 1–13.

[3] André EX Brown and Benjamin De Bivort. "Ethology as a physical science". In: *Nature Physics* 14.7 (2018), pp. 653–657.

[4] Joao Carreira and Andrew Zisserman. "Quo vadis, action recognition? a new model and the kinetics dataset". In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6299–6308.

[5] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[6] Haley L Goodwill et al. "Early life stress leads to sex differences in development of depressive-like outcomes in a mouse model". In: *Neuropsychopharmacology* 44.4 (2019), pp. 711–720.

[7] Lakshmi Narasimhan Govindarajan et al. "Deep spatiotemporal models for behavioral phenotyping of freely behaving mice". In: *(in prep)* (2022).

[8] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[9] Jeffrey W Hofmann et al. "Reduced expression of MYC increases longevity and enhances healthspan". In: *Cell* 160.3 (2015), pp. 477–488.

[10] Hueihan Jhuang et al. "Automated home-cage behavioural phenotyping of mice". In: *Nature communications* 1.1 (2010), pp. 1–10.

[11] Jessy Lauer et al. "Multi-animal pose estimation and tracking with DeepLabCut". In: *BioRxiv* (2021).

[12] Adam Z Lendvai et al. "Analysis of the optimal duration of behavioral observations based on an automated continuous monitoring system in tree swallows (Tachycineta bicolor): is one hour good enough?" In: *PLoS One* 10.11 (2015), e0141194.

[13] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[14] Alexander Mathis et al. "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning". In: *Nature neuroscience* 21.9 (2018), pp. 1281–1289.

[15] Mohammad Sadegh Norouzzadeh et al. "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning". In: *Proceedings of the National Academy of Sciences* 115.25 (2018), E5716–E5725.

[16] Antonio Polino, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization". In: *arXiv preprint arXiv:1802.05668* (2018).

[17] Primoz Ravbar, Kristin Branson, and Julie H Simpson. “An automatic behavior recognition system classifies animal behaviors using movements and their temporal context”. In: *Journal of neuroscience methods* 326 (2019), p. 108352.

[18] Kelsey N Schuch et al. “Discriminating between sleep and exercise-induced fatigue using computer vision and behavioral genetics”. In: *Journal of neurogenetics* 34.3-4 (2020), pp. 453–465.

[19] Cristina Segalin et al. “The Mouse Action Recognition System (MARS) software pipeline for automated analysis of social behaviors in mice”. In: *Elife* 10 (2021), e63720.

[20] Thomas Serre. “Deep learning: the good, the bad, and the ugly”. In: *Annual review of vision science* 5.1 (2019), pp. 399–426.

[21] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[22] Du Tran et al. “A closer look at spatiotemporal convolutions for action recognition”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 6450–6459.

[23] Matthew A White et al. “TDP-43 gains function due to perturbed autoregulation in a Tardbp knock-in mouse model of ALS-FTD”. In: *Nature neuroscience* 21.4 (2018), pp. 552–563.

[24] Ye Emily Wu et al. “Neural control of affiliative touch in prosocial interaction”. In: *Nature* 599.7884 (2021), pp. 262–267.