DeepCell 2.0: Automated cloud deployment of deep learning models for large-scale cellular image analysis

Dylan Bannon[1], Erick Moen[1], Enrico Borba[2], Andrew Ho[2], Isabella Camplisson[1], Brian Chang[1], Erik Osterman[3], William Graf[1], David Van Valen[1, †]

1. Division of Biology and Bioengineering, California Institute of Technology
2. Department of Computer Science, California Institute of Technology
3. Cloud Posse, LLC

†: Corresponding author – vanvalen@caltech.edu

Abstract

Deep learning is transforming the ability of life scientists to extract information from images. While these techniques have superior accuracy in comparison to conventional approaches and enable previously impossible analyses, their unique hardware and software requirements have prevented widespread adoption by life scientists. To meet this need, we have developed DeepCell 2.0, an open source library for training and delivering deep learning models with cloud computing. This library enables users to configure and manage a cloud deployment of DeepCell 2.0 on all commonly used operating systems. Using single-cell segmentation as a use case, we show that users with suitable training data can train models and analyze data with those models through a web interface. We demonstrate that by matching analysis tasks with their hardware requirements, we can efficiently use computational resources in the cloud and scale those resources to meet demand, significantly reducing the time necessary for large-scale image analysis. By reducing the barriers to entry, this work will empower life scientists to apply deep learning methods to their data. A persistent deployment is available at http://www.deepcell.org.

Introduction

Recent advances in imaging – both in optics and in fluorescent probes – are transforming the study of living matter. Thanks to these efforts it is now possible to study cellular function on length scales that span from single molecules[1] to whole organisms[2] with imaging. Concurrent with these advances have been drastic improvements in our ability to computationally extract information from images. Chief among these new tools is deep learning, a set of machine learning tools that can learn effective representation from data in a supervised or unsupervised manner; most successful applications of deep learning have been supervised. These methods have been proven to be more accurate than prior approaches and can automate image classification and image segmentation tasks that have formed the bedrock of single-cell analysis. Their ability to extract latent information from images have also enabled previously unforeseen analyses of cellular function and behavior. Recent applications include interpreting imaging-based screens[3] (image classification), quantifying the behavior of individual immune cells in the tumor microenvironment[4,5] (image segmentation), improving the resolution of images[6] (extracting latent information), and predicting fluorescence images directly from bright field images[7] (extracting latent information).

While deep learning stands to transform our understanding of cellular function through imaging, these tools have yet to see widespread adoption throughout the life scientists. This is surprising, as deep learning tools capable of performing image classification[8], instance segmentation[5,9], and object tracking[10] exist. The barrier impeding such an adoption is two-fold. First, successful deep learning models that can perform a task accurately require a large collection of annotated data for training. Because of this reality, data is just as important to software development as code. Creating a quality dataset for a given task is necessary to apply deep learning but making such a dataset can be both arduous and costly. Increasingly, life scientists are turning to crowdsourcing as a solution to dataset annotation as a solution to this problem[11]. The second aspect is how to deliver deep learning models – both the capacity to train them and the ability to process new data with trained models – to end users. It is this aspect that is the focus of this paper. Unfortunately, there are unique challenges delivering models empowered by deep learning that are both hardware and software in nature. While deep learning has remarkable performance, it is computationally demanding. Most deep learning models have over a million parameters each of which must be tuned during the training process. The practical consequence is that CPUs are often inadequate for deep learning training and inference and specialized hardware accelerators are a necessity. The most common accelerators are NVIDIA graphical processing units[12] (GPUs), although other hardware accelerators such as Google's tensor processing units[13] (TPUs) also exist. The need for hardware acceleration poses a significant financial barrier for labs adopting deep learning methods. Unique software requirements also exist. Python is the most common language used to train and run deep learning models, and often numerous python packages are needed to create an end-to-end image analysis

solution. Proper versioning of these different packages is required for software to be shared among life scientists. Another challenge is that development frequently takes place within a Unix operating system, and considerable Unix system administration experience is necessary to ensure a machine remains stable during model development and deployment.

Cloud computing is the answer to both aspects of the challenges surrounding deployment of deep learning in the laboratory setting. Cloud computing provides the ability to requisition hardware capable of meeting the computational demands of deep learning. Containerization, the encapsulation of software into a container with its computing environment, allows for software to be shipped with proper versioning of dependencies[14]. Furthermore, deployment of software in the cloud also enables the sharing of system administration expertise as a stable deployment only needs to be configured once. Indeed, life science researchers have already begun to turn to the cloud to create and deploy software for cellular image analysis[15,16]. While these prior works applying cloud-based deep learning to cellular image analysis are essential, the full potential of merging cloud computing with deep learning has yet to be realized in this space. Such a merger would remove the barrier hardware and software infrastructure poses to life scientists and pave the way for deep learning-enabled image analysis to become as common place as BLAST searches.

To meet this need, we have developed DeepCell 2.0, an open source framework for training and deploying deep learning models in the cloud. This software grew out of our previous efforts[4,5] applying deep learning to single-cell image analysis and was designed to fully take advantage of cloud computing. Because all the computations take place in the cloud, the software can be run on Windows, Mac, and Unix-based operating systems. We have designed the software to run on both AWS and Google Cloud; users are able to manage their own cloud deployment on either service if they have an account, permission to use the requisite hardware, and sufficient financial resources. A web-based front end enables users to train new models and process data with existing models through a web browser; our software also allows more advanced users to interact directly with the code base through Jupyter notebooks. We have also paid attention to resource allocation – by separating the image analysis pipeline into operations that either require deep learning or do not, we can provide the appropriate compute resources to each operation and significantly reduce operating costs. In addition, our framework can scale compute resources to meet demand, both drastically reducing the time necessary for large-scale image analysis and reducing costs for smaller tasks. Lastly, while this paper is centered around the use case of single-cell image segmentation, we have designed our framework to be flexible enough so that it can be extended as novel applications of deep learning models to biological data arise. A description of the software architecture and the requisite benchmarking is described below.

Software architecture

A full description of our software architecture, and a map of what happens to data – either data to train a model or data to be processed by one – is shown in Figure 1. Here we highlight some of the features of our work that are either enabled by or are unique to cloud computing.

Containerization. To address the issues of creating stability and scalability, we have separated our software into separate modules – kiosk (configures and initiates the cloud deployment), kiosk-autoscaler (scales compute resources), kiosk-frontend (web-based user interface), kiosk-redis (database that manages incoming data), kiosk-redis-consumer (objects that direct data to the appropriate task and perform processing with conventional computer vision operations), kiosk-training (train new models with training data), kiosk-tf-serving (processes data with deep learning models) - that each live within a Docker container. We follow the design rule of one process per container to enhance security, robustness, and debugging.

Infrastructure as code. Cloud computing requires users to specify the configuration of the compute resources requested from the cloud. Details of this configuration for each compute node includes details such as CPU type, memory size, and the presence of a hardware accelerator. The ability to requisition different types of nodes simultaneously provides the opportunity to efficiently match hardware resources with the compute task. While cloud computing platforms provide a user interface for configuring and managing a deployment, we have opted to use an infrastructure as code schema for configuring the compute resources requisitioned from the cloud[17]. This is done by storing the configurations of each type of compute node in yaml files. Users can change details about the

deployment (i.e. what GPU type to use) through a simple terminal menu – these changes are reflected in the updated yaml files. The yaml files are read by the Kubernetes engine to determine what types of compute nodes are available for the cloud deployment. Infrastructure as code also allows us to share all aspects of DeepCell 2.0 – both the source code and the requisite hardware.

Cluster and container orchestration through Kubernetes. We use the Kubernetes engine and its associated package manager Helm to manage both our compute resources and the deployment of the containers that comprise DeepCell. Kubernetes organizes containers into functional units called pods (in our architecture each pod consists of one container), requisitions compute nodes (both with and without GPUs) from the cloud, assigns the appropriate pods to each node, scales the number of nodes to meet demand, and allows for internal communication between containers within the deployment. This architecture enables us to marshal significantly more compute resources for large analysis tasks than is possible from a simple machine image.

Resource allocation. Complete computer vision solutions for cellular image analysis typically require a hybrid of conventional and deep learning methods to achieve a production ready solution[5,18–20]. We have chosen to separate the conventional and deep learning operations so that they run on different nodes – this allows us to use hardware acceleration for deep learning while ensuring conventional operations are only run on less expensive hardware. We have also configured our auto-scaler to shut down GPU nodes if they are idle for an extended period of time.

Testing. Testing is essential for sustainably developing large software projects that make use of cloud computing. We have built unit tests to ensure that future additions to DeepCell preserve existing functionality.

User interface. We have created a simple drag-and-drop user interface from React, Babel, and Webpack. This allows users to both train models and process data through a web browser. Trained models are stored in a cloud bucket where they can be accessed using the predict route. Processed data is downloaded for further analysis in programs like Fiji[21] or in Jupyter notebooks.

Performance Analysis

Models capable of nuclear segmentation in 2D cell culture images[5], 3D confocal images of brain tissue[22], or 2D multiplexed ion beam imaging datasets[4] were used to benchmark DeepCell 2.0. The results of this benchmarking are shown in Figure 2 and demonstrate that DeepCell 2.0 enables rapid and cost-efficient deep learning enabled large-scale image analysis.

Conclusion

While deep learning is transforming the study of living matter, its unique hardware and software infrastructure requirements have been a barrier to its widespread adoption. This work integrates cloud computing with deep learning-enabled image analysis to overcome these barriers and enable life scientists to readily apply these methods to their data.

Data and Source code

All data that was used to generate the figures in this paper are available at http://www.deepcell.org/data.
A persistent deployment of the software described here can be accessed at http://www.deepcell.org. All source code is available at http://www.github.com/vanvalenlab. Detailed instructions are available at http://deepcell.readthedocs.io/.

Acknowledgements

We thank numerous colleagues including Anima Anandkumar, Michael Angelo, Ian Brown, Andrea Butkovic, Long Cai, Markus Covert, Michael Elowitz, Jeremy Freeman, Christopher Frick, Lea Geontoro, KC Huang, Greg Johnson, Leeat Keren, Nora Koe, Takamasa Kudo, Daniel Kyme, Daniel Litovitz, Derek Macklin, Shivam Patel, Nicolas Pelaez Restrepo and Cole Pavelchek for helpful discussions and gratefully providing the data necessary to develop this work. We gratefully acknowledge support from the Allen Discovery Center at Stanford University, Google Research Cloud, Figure 8's AI for everyone award, and a subaward from NIH U24CA224309-01.

References

1. Liu, H. *et al.* Visualizing long-term single-molecule dynamics in vivo by stochastic protein labeling. *Proc. Natl. Acad. Sci. U. S. A.* **115**, 343–348 (2018).
2. McDole, K. *et al. In Toto* Imaging and Reconstruction of Post-Implantation Mouse Development at the Single-Cell Level. *Cell* **175**, 1–18 (2018).
3. Kraus, O. Z. *et al.* Automated analysis of high-content microscopy data with deep learning. *Mol. Syst. Biol.* **13**, 924 (2017).
4. Keren, L. *et al.* A Structured Tumor-Immune Microenvironment in Triple Negative Breast Cancer Revealed by Multiplexed Ion Beam Imaging. *Cell* **174**, 1373-1387.e19 (2018).
5. Van Valen, D. A. *et al.* Deep Learning Automates the Quantitative Analysis of Individual Cells in Live-Cell Imaging Experiments. *PLOS Comput. Biol.* **12**, e1005177 (2016).
6. Weigert, M. *et al.* Content-Aware Image Restoration: Pushing the Limits of Fluorescence Microscopy. (2018).
7. Ounkomol, C., Seshamani, S., Maleckar, M. M., Collman, F. & Johnson, G. R. Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy. *Nat. Methods* 1 (2018). doi:10.1038/s41592-018-0111-2
8. Hughes, A. J. *et al.* Quanti.us: a tool for rapid, flexible, crowd-based annotation of images. *Nat. Methods* **15**, 587–590 (2018).
9. Owens, J. D. *et al.* GPU Computing. *Proc. IEEE* **96**, 879–899 (2008).
10. Jouppi, N. P. *et al.* In-datacenter performance analysis of a tensor processing unit. in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* 1–12 (2017). doi:10.1145/3079856.3080246
11. Merkel, D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J* **2014**, (2014).
12. Haberl, M. G. *et al.* CDeep3M—Plug-and-Play cloud-based deep learning for image segmentation. *Nat. Methods* **15**, 677–680 (2018).
13. McQuin, C. *et al.* CellProfiler 3.0: Next-generation image processing for biology. *PLOS Biol.* **16**, e2005970 (2018).
14. Walt, S. van der *et al.* scikit-image: image processing in Python. *PeerJ* **2**, e453 (2014).
15. Bai, M. & Urtasun, R. Deep Watershed Transform for Instance Segmentation. in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2858–2866 (2017). doi:10.1109/CVPR.2017.305
16. Wang, W. *et al.* Learn to segment single cells with deep distance estimator and deep cell detector. *arXiv* (2018).
17. Abramoff, M.D., Magalhães, Paulo J. & Ram, Sunanda J. Image processing with ImageJ. *Biophotonics Int.* **11**, 36null (2004).
18. Shah, S., Lubeck, E., Zhou, W. & Cai, L. In Situ Transcription Profiling of Single Cells Reveals Spatial Organization of Cells in the Mouse Hippocampus. *Neuron* **92**, 342–357 (2016).

Figures

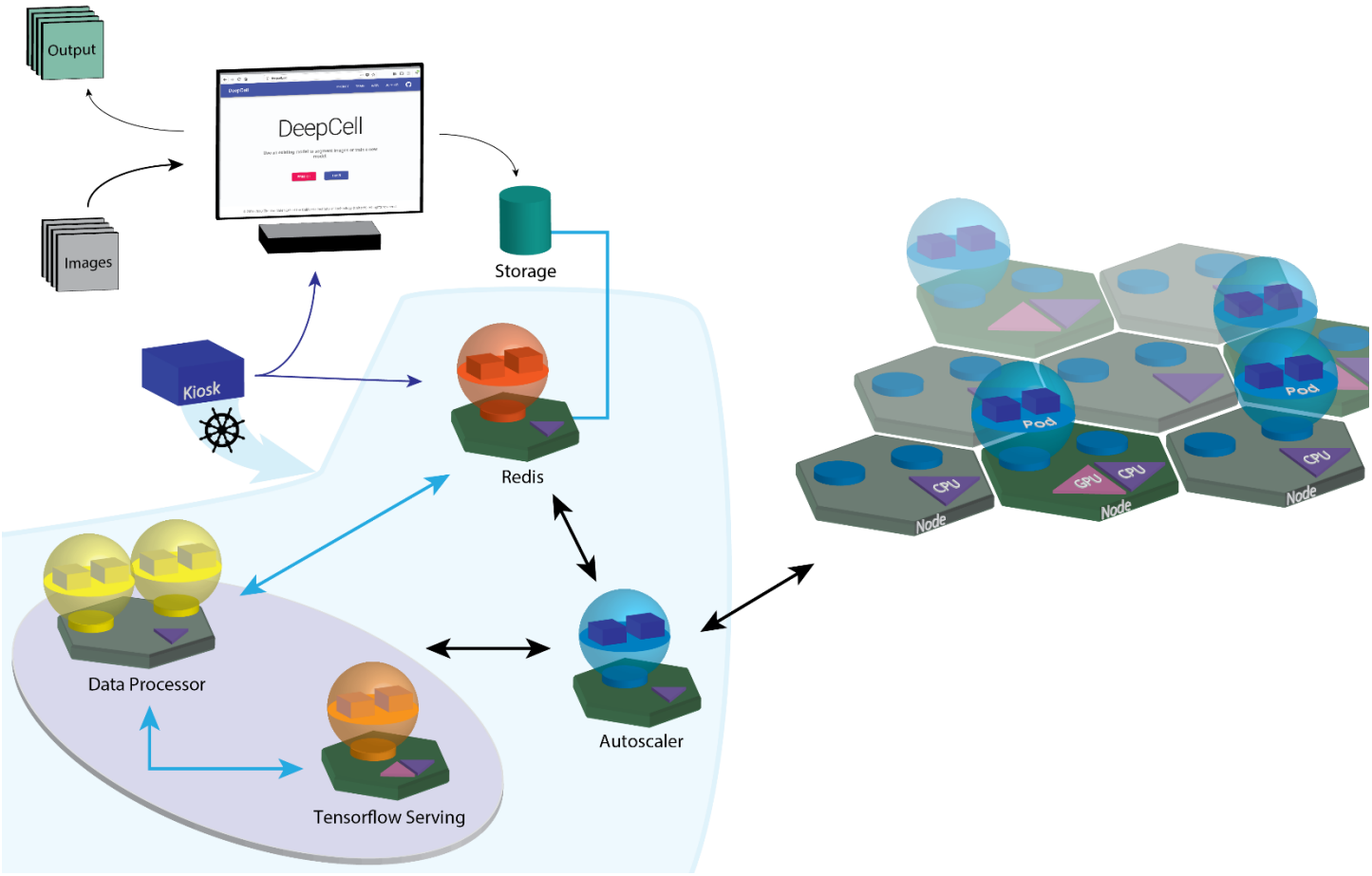

Figure 1: Software architecture of DeepCell 2.0. Blue lines signify flow of data while black lines signify flow of compute resources. Users begin by starting the kiosk container and specifying parameters (user credentials, GPU type, etc.) of the cloud deployment. These parameters are used by Kubernetes to construct the cloud deployment and assign pods (collections of containers) to their appropriate node. Users submit data for either processing or training through their web browser. The web interface allows users to select parameters for training feature-nets – deep learning models designed for general purpose instance segmentation[5] – and lets users choose an image analysis workflow to process data. The image analysis workflow consists of pre and post processing operations and a deep learning model (which exists in a cloud bucket). Uploaded data is placed in a cloud bucket and triggers the creation of an entry into a Redis database. A data processor communicates with the database and coordinates the shuttling of data. Data submitted to train a deep learning model initiates the creation of a training job within the Kubernetes engine. Training progress can be monitored through Tensorboard. Data submitted for processing is subjected to a combination of deep learning (provided via Tensorflow Serving) and conventional computer vision operations. Trained models are stored in a cloud bucket and are then made available through the prediction interface. Processed data is downloaded for further analysis. An autoscaling module queries the Redis database to monitor demand and increases or decreases the number of compute nodes and active pods as needed.

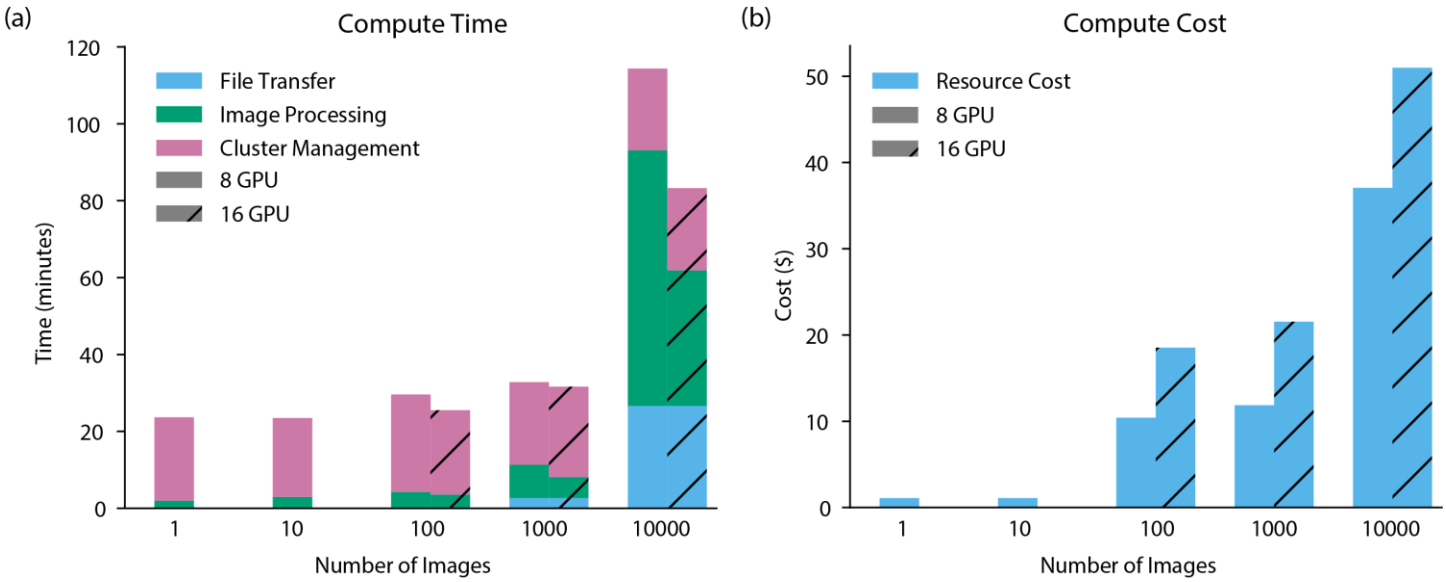

Figure 2: Benchmarking of DeepCell 2.0. The autoscaler was allowed to requisition a maximum of 8 or 16 GPUs for image processing during benchmarking. (a) Processing time as a function of image batch size. The amount of time required for file transfer, cluster management (creation and destruction), and image processing is shown. (b) Cost as a function of image batch size. By efficiently allocating computational resources, DeepCell 2.0 is a cost-effective approach to large-scale image analysis. Costs were computed by weighting instance run times by the spot pricing on Google Cloud.