# Cell Segmentation on the Edge for Diagnostic Microscopy

**Alexander Conklin**
Department of Applied Physics
Stanford University
conklin1@stanford.edu

**Kevin Marx**
Electrical Engineering
Stanford University
kmarx@stanford.edu

**Ethan Hellman**
Computer Science
Stanford University
hellman1@stanford.edu

**Rehaan Ahmad**
Computer Science
Stanford University
rehaan@stanford.edu

## Abstract

Microscopy is a powerful tool for diagnostics of bloodborne pathogens. Recent advances in the development of low cost light microscopes for imaging blood smears hold the promise of bringing new diagnostic methods to remote and economically disadvantageous regions of the world. For example, testing for malaria using an enzyme-linked immunosorbent assay (ELISA) is costly and requires a fresh supply of reagents while blood imaging could provide diagnoses at lower cost and higher throughput. However, in order to enable to full promise of remote blood imaging, we must develop efficient image processing pipelines. To make the technology as accessible as possible, the image processing pipeline must be run on affordable hardware, operate without access to the internet, and consume no more than 10 W of power during its operation. In this paper we adapt state of the art deep learning models for cell segmentation through model adaptation, quantization, and the use of various graph compilers. We show these techniques can yield competitive performance, as measured by the Jaccard Similarity Index, underscoring the suitability of edge processors for edge blood diagnostics.

## 1 Introduction

The World Health Organization reports 274 million cases of malaria globally in 2021 with an estimated 619 thousand deaths [2]. Malaria is curable and these deaths were preventable had there been a rigorous diagnostics and healthcare system. Malaria is most prevalent in equatorial regions and its range intersects with several developing countries that lack the electrical infrastructure, internet connectivity, and healthcare system necessary to diagnose and treat all those who are infected [1, 13]. Decreasing the resources necessary to run malaria diagnostics and increasing diagnostic throughput is an important step in reducing the amount of malaria-related deaths globally.

EE292D: Machine Learning on Embedded Systems 2022, Stanford University

There are two approaches for detecting malaria, chemically detecting the presence of antibodies and visually detecting the parasites under a microscope. Chemical detection is difficult because the reagents degrade over time and require significant amounts of time, energy, and equipment to synthesize on site [16]. On the other hand, visually detecting parasites is bottlenecked by having a trained expert manually inspect blood smears for the signs of malaria. Machine learning on edge devices can automate this process at low cost and power consumption and have the potential to greatly increase the throughput of diagnostic systems without relying on an internet connection.

Rather than creating a model for detecting malaria directly, we are first creating a model for segmenting cells in view. Cell segmentation, the process of masking each cell in an image and giving each a unique label, is a useful first step in many cell analysis pipelines. Segmentation also makes generating training data and verification easier; rather than manually drawing circles around each malaria parasite in a photo and comparing it to what the computer infers, one instead can mark cells as either containing a parasite or not containing a parasite.

Our methodology was to first acquire valid training data for segmented red blood cells (RBCs). Next, we decided on which hardware to use for inference. We then compared different models for cell segmentation that might perform well on edge devices. Once we decided on what model to use, we investigated methods for quantizing the model and deploying it on an edge device.

## 2 Background

### 2.1 Octopi Microscope

Octopi is an open configurable high-throughput imaging platform for disease diagnosis on the edge [10]. The microscope, design for a resource-constrainted setting, makes use of two main magnification modes: a low magnification mode using re-purposed cellphone lenses for the objective, and a high magnification mode using a standard microscope objective. A low cost CMOS sensor, the Pi Camera, is used for imaging. Other cost cutting measures include a low cost actuator for mechanical stage translations, LEDs in the illumination modules, direct diode lasers and diode-pumped lasers (found in common house hold electronics) for fluorescence excitation, and a Raspberry Pi single board computer (SBC) for control of the microscope. Ultimately, the CMOS sensor connects to the SBC via the Camera Serial Interface for encoding and further processing. For further details we refer the reader to the Octopi paper:[10]. For our project, we are using images taken by the Octopi microscope to demonstrate it is a viable tool for malaria diagnostics on the edge.

### 2.2 Neural Networks for Cell Segmentation

The key steps of medical imaging include image capturing, image encoding, image processing and image analysis (post-processing). Advances in imaging technology paired with digital signal processing (DSP) hardware has increased the rate at which we can create images, making image processing the primary bottleneck in the pipeline. In many medical applications, image processing translates consists of segmenting the image into regions of interest. For example, one might want to segment an image of cells into individual cells and flag each cell with a particular class. Simple cell segmentation is the foundation of more advanced semantic segmentation methods and is an ideal preliminary task to target.

Historically, cell segmentation has been done by hand, requiring hours of work and limiting applications to the order of thousands of cells rather than millions. The advent of more powerful CPUs and GPUs led to the use of standard machine learning techniques such as k-clustering and support vector machines to perform this analysis. These methods have shown modest success when the task can be crafted into a classification problem but come at the cost of requiring manual data massaging and do not generalize well to other images or classification problems[8].

More recently, neural networks have shown promising results in an array of segmentation and semantic segmentation results [14]. Early models employed convolutional neural network (CNN)-based architecture but suffered poor output quality, with reconstructed image lines failing to close. A germinal work showed residual connections are crucial to recovering spatial information when reconstructing the output image, significantly improving accuracy and reducing the vanishing gradients problem [7]. Today's state of the art models, such as Cellpose, share a few common attributes including a general u-net structure (inspired by generative models) with residual connections [15].

While Cellpose exhibits a degree of generalizability, and routinely hits intersection-over-union (IOU) scores over 0.90 in segmentation tasks, the model makes use of 3D convolutions and is out of reach for the computational budget (< 10 Watts) of edge systems.

## 3 Dataset

The Prakash Lab tests its equipment in the field and recently acquired a large dataset red blood cell images from Tanzania. RBCs are flat and relatively hard to image due to poor contrast. However, by combining a flatfield image of the cells with an image of the cells illuminated from the left and illuminated from the right, one can generate cell images with significantly improved contrast. Once the microscope is set up, it will consistently create contrast-enhanced images but any differences in how its lights are set up will cause its images to look slightly different from the images generated by other microscopes. Instead of using a large model that can handle every lighting configuration, we instead are using edge learning so the same low-cost model can learn to segment images created by a given microscope.

Our initial training images are generated using the differential phase contrast algorithm, shown in Fig. 1. The data is augmented by flipping it vertically and horizontally and by taking random crops of the full images. We avoid rotating the images because the data does have a preferred orientation given by the positioning of the left and right lights.

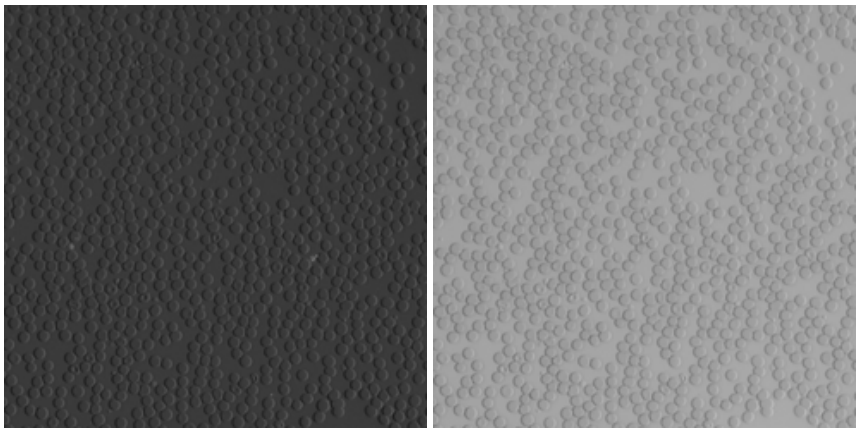

Figure 1: Original Differential Phase Contrast (DPC) image (left) and "occluded" DPC image (right).

We then trained a preliminary model on the unmodified data. Once we confirmed the model works, we then modified the differential phase contrast algorithm until the preliminary model was no longer able to segment the cells properly but still visually appears to have divisions between the cells, as see in in Fig. 1. This modified data then is used for edge training to demonstrate the model can learn to adapt to microscopes with suboptimal lighting conditions.

## 4 Methods

### 4.1 Model Overview

We first selected a segmentation model that struck a good balance of performance and resource usage: Cellpose, a generalist 3D U-net; U-net, a 2D u-net; ResnetFCN, a modified residual network; and m2u-net, a modified u-net employing the encoder used by mobilenetv2 [15, 12, 4, 9]. These models were benchedmarked in FP32 implementations as a base test for further exploration and modification.

### 4.2 Quantization Methods

As part of the Nvidia family, the Jeston Nano architecture makes use of CUDA cores for tensor processing. These cores are reconfigurable, supporting lower precision FP16 and INT8 matrix-vector operations at higher speed and lower resource utilization. The Jetson's reconfigurable architecture

and flexibility let us survey several state of the art quantization and deployment methods, including post training quantization and quantization aware training [11]. We initially considered ZeroQ (post-training quantization) and HawQV3 (quantization aware training), two recent academic frameworks which achieve superb performance and flexible quantization schemes (layer wise) down to INT4 [6, 17]. Unfortunately, applying these quanitzation tools was not trivial and while the excelled for some of models in the PytorchCV Zoo, they were not easily amenable to the netlist structure used in our torch models and were missing for certain operators. Additionally, the Jetson Nano doesn't natively support arbitrary precision modes and could not run sub INT8 precision on individual layers.

Without the need to explore sub INT8 quantization we deferred to the standard Torch quantization toolbox. We modified our network to remove ReLU6 in favor of ReLU activations and we modified the neural network netlist to omit concatenation operators from quantization in order to support the methods. For post training quantization, we employed static activation quantization by feeding the model the test dataset to gather activation distributions.

### 4.3 Deployment

Figure 2 depicts a flow chart of blood cell image capturing and processing with the Octopi microscope and Jetson Nano, with associated estimates of latency for each $3kx3k$ image. The latency of each step contextualizes the performance we obtain in image processing - sub 100 ms inference on the full resolution image would be great, the gains diminish as preprocessing becomes a bottleneck. We narrow the scope of this project to accelerating the image processing on the Jetson, comparing the Pytorch API with TensorRT for model deployment.

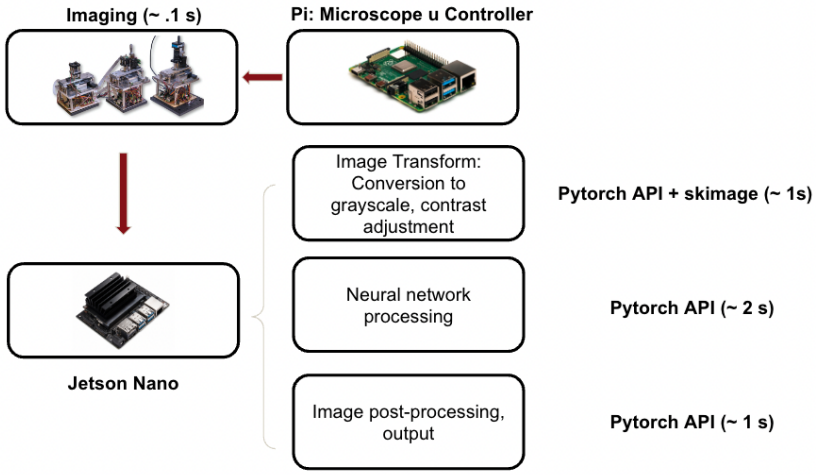


Figure 2: Flow chart of blood sample processing and processing.

### 4.4 Edge Tuning

One of the main motivations for this project was to create an adaptive model. Given that imagery from different microscopes from different parts of the world may be slightly different due to lighting, cell population, etc., our aim was to determine the most optimal training schedule for fine-tuning our UNet architecture. Limited research has been done on fine-tuning specifically UNet architectures. Amiri et al. [3] tested the effect of various different fine-tuning schedules for UNets and found performant schedules for these models. While prevailing knowledge would suggest that keeping as many layers intact while fine-tuning only deeper layers of a network will result in greatest improvements [5], Amiri et al. showed how freezing different combinations of deep and shallow layers as well as contracting and expanding components of a UNet architecture can be effect fine-tuning strategies.

The main objective of our work is to determine what fine-tuning technique can be applied to our M2Unet model to reduce the number of total trainable parameters and training time while maximizing the overall performance of the model. Given the differences between the M2Unet architecture and pretraining data used in this experiment compared to that in Amiri et al., we chose to perform simpler

experiments. More specifically, we tested 6 different types of training strategies to determine whether fine-tuning the encoding, decoding, contracting, expanding, or combination of layers would result in performance gains as demonstrated in Fig. 4. To reduce overall amount of training on the slower edge system, the fine-tuning tests were conducted on the cloud to determine the most promising strategies. Intuitively, performance increases that may be observed on the cloud would necessarily translate to the edge. From there, similar tests would be run on edge using a smaller set of top strategies.

In order to simulate different training environments, the original data was augmented to introduce random noise. Our baseline, unquantized M2Unet architecture was then loaded. According to different strategies, a variable number of layers within the network were frozen. Fine-tuning was then done for 5 epochs on the new data. Training and inference times were gathered for each model as well as the Jaccard score based off of final prediction and final prediction probabilities. Using these metrics, we can observe how computational cost of fine-tuning changes according to different schedules along with model confidence and accuracy.

## 5 Results

Table 1 depicts the performance of the four segmentation models we evaluated. Leveraging the first 14 layers of the mobilenetV2 design, M2u-net implements an efficient encoder and upsampling method, with the smallest memory footprint (not critical for the Jetson Nano but important for other edge options) and fastest inference and training (single forward + backward pass) across the board. All other models had an order of magnitude increase in the latency with the exception of the u-net architecture which was only slightly slower but came with a 60x memory footprint. As such, we elected the M2u-net for further investigations.

M2u-net was training over our restricted dataset, using random transformations to extend the the data, reflecting real world constraints of limited labeled data. The best of 5 randomly seeded models exhibited a Jaccard score of .9791, a measure of similarity defined by the size of the intersection of two shapes divided by the size of the union, competitive with state of the art models (Table 2). While this is impressive, we temper results by noting Cellpose achieves this score on a variety of more complex semantic segmentation models. Figure 3 depicts the sample output of the network before the post-processing greyscale threshold used to make the output more useful to the human user. The model picks up minimal noise in the output image, and shows crisp edges around cells, even showing texture lines not visible to the human eye in the input image.

Quantization to FP16 is done by direct casting and shows no significant accuracy loss, while exhibiting a .015 s speed up. Post training static quantization to INT8 with activation sampling from the test set resulted in rather negligible drops in accuracy, decrease the Jaccard score by .05 points 2. Notably the compressed model is unable to compile and crashes the kernel with run with CUDA, demonstrating bugs in our deployment tool. In the CPU setting the model performs worse by a factor of 3x, as the CPU implements a psuedo-quantization incurring an overhead for rounding, quantization and dequantization - it is not able to benefit from using INT8 operators.

In order to fully utilize quantized aware training and post training quantization to INT8 we set our sights on using TensorRT to deploy the model. Even for the the FP32 case, TensorRT exhibits about a 2x improvement over the native Torch API. The quantized M2unet was converted to ONNX, an intermediate representation that is optimized to run efficiently with the TensorRT engine. Ultimately we failed to parse the ONNX model and create a TensorRT engine - this is attributed to the fact our model netlist used upsampling without constant scales. This operator is not yet supported, preventing our implementation of the model from running with TensorRT.

Our fine-tuning results gave limited insight into the utility of fine-tuning on the edge. Fig. 5 shows how computation time for inference is largely and intuitively unaffected. This is to be expected given that the computation graph remains unchanged for each model. The training time, however, does increase depending on the training schedule. Fig. 6 demonstrates how only fine-tuning Schedule 2 (only fine-tuning the decoder) remained unchanged in training time. All other schedules increased in training time.

In terms of performance, schedule to schedule, this was largely unaffected. Observing Fig. 7, Schedule 4 is the only fine-tuning schedule that seems as though it may have benefited from more training. Schedule 6, however, appears to have severely decreased in performance as a result of

| Model | Parameters (Millions) | Jetson FP32 Inference Latency (batch size = 1) | Jetson FP32 Training Latency (batch size = 1) | Comments |
|---|---|---|---|---|
| Cellpose | > 30 M | > .5 s | > 2 s | Too slow and excessive for our narrow task; high memory footprint |
| U-net | 30 M | .12 s | > .41 s | Large model but relatively fast; high memory footprint |
| M2u-net | .55 M | > .055 s | > .195 s | Smallest memory footprint; very fast inference and backwards path |
| Resnet50-FCN | 28 M | > .51 s | > 1.48 s | Slow and high memory footprint |

Table 1: Comparison of four segmentation/semantic segmentation models deployed in FP32 for batch one inference and training on the Jetson Nano. Benchmark uses 512 x 512 input image.

| Model | Quantization | Jaccard Score | Jetson CPU Inference Latency (batch size = 1) | Jetson CUDA Inference Latency (batch size = 1) |
|---|---|---|---|---|
| M2u-net | INT8 | .08745 | .18 s | Not supported |
| M2u-net | FP32 | .09791 | .058 s | .055 s |
| M2u-net | FP16 | .09756 | .61 s | .039 s |

Table 2: Static post training quantization accuracy loss and latency comparison to FP32 when model run on different processing cores. All weights and activation are in INT8.

increased fine-tuning. Lastly, we observed changes in confidence intervals based on fine-tuning schedule. In order to measure this, we observed the maximum and maximum confidence observed for a given schedule for a certain number of layers, and took the difference. Ideally, we would like to see the confidence interval decrease as predictions are made with predictable certainty. Looking at Fig. 8, only Schedule 2 observed a notable decrease in confidence interval while all others stayed relatively consistent.

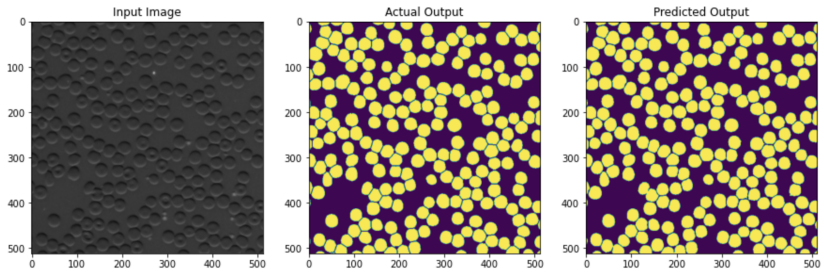


Figure 3: Results of M2UNet Inference

## 6 Discussion

Broadly our results show without question the M2u-net can be run efficiently on edge hardware for cell segmentation. Our methods of extending limited datasets through random crops not only reflect real world constraints but proved fruitful for training. The network is robust to static post training quantization down to INT8 showing only minor accuracy drops. While the speeds achieved by using CUDA in the Torch API were impressive there is more room for acceleration using TensorRT. To this end, rewriting our model in tensorflow would provide an avenue to truly realize the INT8 acceleration in quantization and ought to be an area for future exploration. While our survey of novel quantization techniques revealed some interesting approaches, our experimentation showed these frameworks are limited in the operators and netlists supported and not representative of our target hardware. Practically, built in methods of static post training quantization and quantization aware training with FP32 backwards pass are sufficient.

Training on the edge proved difficult, but our measurement of a forwards and backwards pass in FP32 as .195 s in allows training of 18,000 512 x 512 images per hour. This number is slowed to 6,000 when accounting for the Jetson Nano's limited RAM and having to create the training samples with random transformations and discard them during training. This speed is sufficient to support the goal of training M2u-net on the edge overnight as might be desired in a remote deployment of the microscope. ONNX, TVM, and other frameworks are designed for speeding up inference and it will be difficult to run efficient training on the edge.

Our fine-tuning results did not provide the greatest insight into the potential benefits of fine-tuning our M2UNet architecture. Intuitively, we saw that there was not much change in inference time according to different fine-tuning schedules. This is because the final computation graph will remain the same. More interestingly, however, is the observation that different fine-tuning schedules have

implications on the training time. Furthermore, through only fine-tuning on the decoder, the training time can be kept constant for the M2Unet.

Moreover, we observed interesting performance trends. While no schedule clearly showed increases in performance as a result of the schedule and the number of fine-tuned layers, Schedule 6 clearly showed a dramatic decrease in performance. Common practice suggests that training the later layers of a pre-trained network works best for fine-tuning; and schedule 6 was trained by fine-tuning the initial layers of the encoder and decoder. As such, it is logical that this could drastically disrupt the overall performance of the model. Lastly, as we observed with Fig. 8, Schedule 2 was the only schedule that demonstrated a decrease in confidence interval. While a crude metric, this may suggest that the model is reaching a more stable state of confidence. We regard this to be a good thing. Rather than have some highly confident predictions and some low predictions, we would rather have less variable predictions that are ideally higher. Fig. 8 may suggest that Schedule 2 is trending towards this stability.

## 7 Conclusion/Future Work

In this work we put forth a machine learning based framework that can adapt to new onsite data and produce satisfactory results when measured by the Jaccard Similarity Index. The device satisfies our initial goal performing this edge computation with under 10 W of power. With the power (and thus compute) constraints imposed for this situation, we implement models that are robust enough to solve the relatively difficult problem of cell segmentation but are also lightweight enough to execute on the Nvidia Jetson Nano edge device. In a real world setup for this problem, data is directly obtained from an inexpensive microscope. This edge fine-tuning pipeline is replicated in our process by holding out sets of separate cell data with slight distributional shifts. While the M2Unet based approach performs well in the standard training/testing ML setup, there is still room for improvement with the model's edge-training performance on the augmented datasets meant to mimic onsite microscope data. While quantization to int8 was not necessary for successful M2Unet inference on the Jetson, we observed that using simple post training quantization techniques did not yield useful results when compared to the FP32. In general we were successfully able to train models to hit .97 Jaccard score in FP32. To our knowledge, this is the first attempt at deploying, fine-tuning, and quantizing a UNet architecture on the edge. Future work should focus on quantized-aware training on the edge.

Our code is available on `https://github.com/kmarx-kmarx-kmarx/m2unet_quantized`.

## References


[1] Cdc - malaria - about malaria. `https://www.cdc.gov/malaria/about/`, 2022.

[2] Fact sheet about malaria. `https://www.who.int/news-room/fact-sheets/detail/malaria`, 2022.

[3] Mina Amiri, Rupert Brooks, and Hassan Rivaz. Fine tuning u-net for ultrasound image segmentation: which layers? 2020.

[4] Lei Bi, Dagan Feng, and Jinman Kim. Dual-path adversarial learning for fully convolutional network (fcn)-based medical image segmentation. *The Visual Computer*, 34(6):1043–1052, 2018.

[5] Oscar Beijbom Judy Hoffman Trevor Darrell Brian Chu, Vashisht Madhavan. Best practices for fine-tuning visual classifiers to new domains. 2016.

[6] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

[7] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. In *Deep learning and data labeling for medical applications*, pages 179–187. Springer, 2016.

[8] Intisar Rizwan I Haque and Jeremiah Neubert. Deep learning approaches to biomedical image segmentation. *Informatics in Medicine Unlocked*, 18:100297, 2020.

[9] Tim Laibacher, Tillman Weyde, and Sepehr Jalali. M2u-net: Effective and efficient retinal vessel segmentation for resource-constrained environments. *arXiv preprint arXiv:1811.07738*, 2018.

[10] Hongquan Li, Hazel Soto-Montoya, Maxime Voisin, Lucas Fuentes Valenzuela, and Manu Prakash. Octopi: Open configurable high-throughput imaging platform for infectious disease diagnosis in the field. *BioRxiv*, page 684423, 2019.

[11] Mariam Rakka, Mohammed E Fouda, Pramod Khargonekar, and Fadi Kurdahi. Mixed-precision neural networks: A survey. *arXiv preprint arXiv:2208.06064*, 2022.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[13] Nick Routley. Mapped: The 1.2 billion people without access to electricity. 2019.

[14] Hyunseok Seo, Masoud Badiei Khuzani, Varun Vasudevan, Charles Huang, Hongyi Ren, Ruoxiu Xiao, Xiao Jia, and Lei Xing. Machine learning techniques for biomedical image segmentation: an overview of technical aspects and introduction to state-of-art applications. *Medical physics*, 47(5):e148–e167, 2020.

[15] Carsen Stringer, Tim Wang, Michalis Michaelos, and Marius Pachitariu. Cellpose: a generalist algorithm for cellular segmentation. *Nature methods*, 18(1):100–106, 2021.

[16] Lotus L. van den Hoogen et. al. Comparison of commercial elisa kits to confirm the absence of transmission in malaria elimination settings. *Frontiers in Public Health*, 2020.

[17] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021.

# 8 Appendix

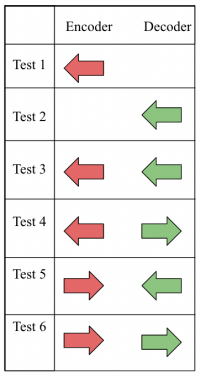


Figure 4: Fine-tuned layers for a given schedule

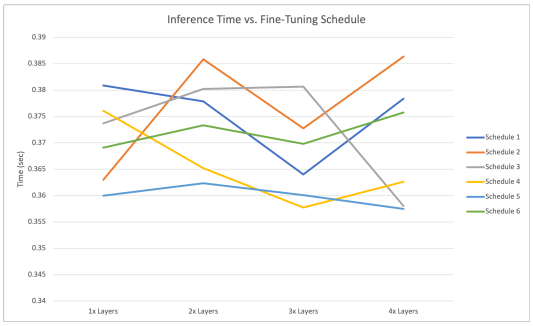


Figure 5: Inference time for given fine-tuning schedule and number of layers

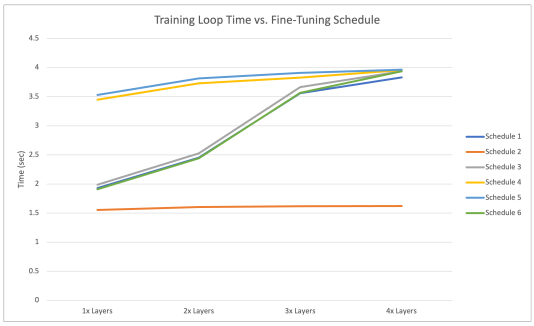


Figure 6: Training time for given fine-tuning schedule and number of layers

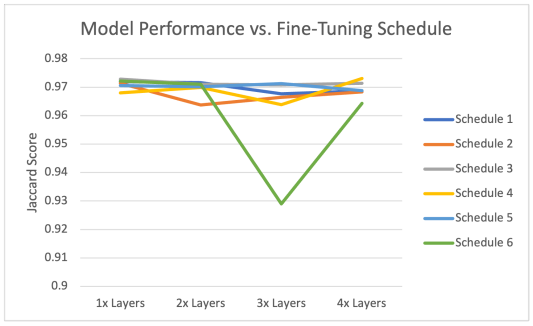


Figure 7: Performance (Jaccard score) for given training schedule and number of layers

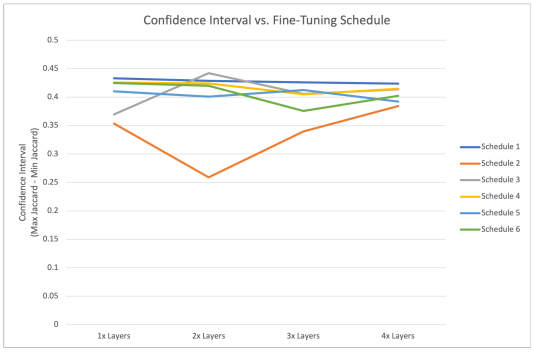


Figure 8: Training time for given fine-tuning schedule