A Functional Tissue Unit Segmentation Program for Histology

Wren McQueary
Department of Computer Science
George Mason University
Email: wmcquear@gmu.edu

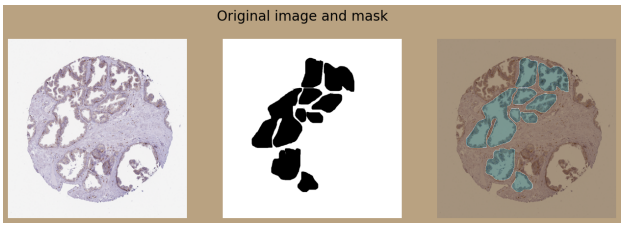

Fig. 1. An example of a ground-truth FTU segmentation. Original figure credit: [1]

***Abstract*—Kaggle's *HuBMAP + HPA - Hacking the Human Body* competition involves segmenting functional tissue units (FTUs) in images of human tissue microscope slides. The top 5 submissions to the project finetune a pretrained UNet model. I investigated whether a DeepLabV3 model might perform better at the task, since this newer model is designed to be particularly robust to changes in scale, which are common in microscopy. However, DeepLabV3 proved to be unsuitable. Even after multiple rounds of grid searching and adjusting training parameters, DeepLabV3 failed to learn the segmentation task effectively. By contrast, I changed the model to UNet and trained it under almost exactly the same circumstances, and it obtained an accuracy in the top 5 of Kaggle submissions on the first attempt. This project constitutes a more intensive investigation of deep image segmentation than our brief brush with it in class.**

I. Introduction

The Kaggle competition *HuBMAP + HPA - Hacking the Human Body* is a segmentation task for functional tissue units (FTUs), which are contiguous groups of cells that perform a shared purpose. In the competition, FTUs must be segmented, given a microscope slide. An example of this segmentation task is shown in Figure 1. For my project, I built a deep image segmentation model to attempt the Kaggle challenge.

Microscope image segmentation tasks are important in fields such as medicine, biology, and public health [2]. This Kaggle competition is representative of such a task, and shares many of the same challenges. Robustness to organ type is important, as the challenge involves cross-sections from five different organs. Another challenge is that the images in the dataset vary wildly in resolution, from as little as 160x160 pixels to as much as 4500x4500 pixels. Also, since these images were obtained using microscopes, they may vary significantly in magnification level [3].

In light of these challenges, any solution needs to be robust to large disparities in image scale. Therefore I chose to finetune a pretrained DeepLabV3 model [4]. DeepLabV3's defining characteristic is that its forward pass involves multiple parallel runs with different atrous dilation factors and spatial pyramid pooling, to improve the model's robustness to changes in object scale. I sought to investigate whether this robustness would make DeepLabV3 a more effective solution than UNet, which is used in all top 5 Kaggle submissions for the challenge yet lacks this feature and is two years older [5] [6]. Of the two DeepLabV3 architectures available in PyTorch (ResNet50 and ResNet101), I chose to use ResNet101 because of its greater expressiveness [7]. In this task, expressiveness is an important trait, because even humans can struggle to discern between FTUs and other collections of cells.

My project expands on our course's light coverage of segmentation. In class, we only briefly laid out the differences between the goals of semantic segmentation, image classification, object detection, and instance segmentation. Other than that comparison, only shallow approaches were explored in-depth for semantic segmentation, such as graph-based methods, k-means clustering, and the mean-shift algorithm. In my project, I explore deep segmentation more thoroughly by implementing my own image segmentation tool using PyTorch [7].

I found that DeepLabV3 is an ineffective tool for this problem. My best DeepLabV3 model for this task attained an accuracy of only 9.57%, despite multiple rounds of grid search and training parameter adjustment. By contrast, I also trained my own UNet model and found that UNet performs significantly better than DeepLabV3. With no alterations from the DeepLabV3 setup (except for an untuned learning rate change), my UNet model's accuracy is 87.09%, which falls within the same range as the top 5 submissions to the original Kaggle competition [6][3].

II. Approach

When initially choosing a pretrained model, I considered using DeepLabV3 [4], an FCN model [8], and LRASPP [9]. I eliminated LRASPP because its design is tailored for video rather than images, making it unsuitable for this task. I eliminated FCN for two reasons: (1) it performs worse than DeepLabV3 on the PASCAL VOC 2012 test set (with an

accuracy of 79.1% to DeepLabV3's 85.7%), and (2) it lacks any feature similar to DeepLabV3's parallel atrous passes.

A. Creating the Jupyter Notebook

My Jupyter Notebook uses random seeding to aid reproducibility. All my work was performed with a random seed of 0. I wrote a custom function, `seed_all()`, which seeds `random`, `numpy`, and `torch` all with the same seed.

To load the dataset provided by Kaggle, I adapted code from another Kaggle competitor, Zakaria Joudar [10], which loads images into OpenCV2 so they can easily be passed off to PyTorch for conversion into tensors. I wrote a custom PyTorch Dataset object to hold these images and apply normalization and image augmentation. Image augmentation consisted of random sharpness adjustment (to approximate errors in microscope focus) and color jitter (to approximate different staining techniques).

In the dataset as-given, the segmentation labels are expressed as run-length encodings (RLEs), rather than full tensors. For better integration with PyTorch, I needed to convert these labels to tensors with the same shape as the image they labeled. To achieve this, I borrowed the function `rle_to_mask()` from another Kaggle competitor, Hyunwoo Lee [11].

For the loss function, I used Dice loss, the same function that is used for scoring the competition [3]. Dice loss is expressed as a value in the range $[0, 1]$, and is defined as $1 - 2\frac{y \cap y_{pred}}{y + y_{pred}}$, where the numerator is the intersection between the true and predicted segmentation, and the denominator is their combined magnitude. Dice accuracy, also called the dice coefficient, is expressed as 1 minus the dice loss. For my optimizer, I initially used Adam, but later switched to Ranger to improve my models' accuracies, as discussed in Section II-B2 [12].

DeepLabV3 outputs 21 tensors when given an image: background, aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, and tvmonitor. By contrast, the Kaggle task requires only one semantic label. Therefore I chose to train and use only one of DeepLabV3's 21 output tensors. I chose to use the `cat` segmentation tensor, because although none of the 20 non-background classes resemble FTUs' combination of amorphousness and multicoloredness, cats are subjectively the closest to having these traits. Figure 2 shows a variety of cat images from ImageNet, on which DeepLabV3 is trained. Note the variety in these cats' silhouettes, how blob-like many of them are, and the fact that many have dappled patterns reminiscent of FTUs.

I chose to scale all images to a common size. Many of the microscope slide images are quite large, with a maximum resolution of 4500x4500 pixels. My computer's 16 GB of GPU VRAM cannot hold an image tensor of this size. Furthermore, due an issue with PyTorch's implementation of `nn.BatchNorm`, each training batch must contain at least 2 images [14]. Therefore I chose the maximum possible standard resolution at which two images could be held in my VRAM: 512x512 pixels.

Fig. 2. A variety of ImageNet images with the label `cat`. Original figure credit: [13]

The aforementioned requirement that each training batch contain at least 2 samples also informed my data split. I split my 351 images into a training set of 256 images and 95 validation images. By having the number of training images be a power of 2, I ensured that each batch would always contain at least 2 images, as long as I set my batch size to a power of 2 as well. As explained later in Section II-B2, I later increased the size of my dataset to 1024 training samples and 207 validation samples.

To automate grid search, I built a Jupyter notebook in which an entire round of training is contained within a function called `perform_training()`. This function can then be called in a `for` loop to run a full round of training for each desired combination of hyperparameters. The `perform_training()` function also automatically saves the best model from each round of training (as determined by validation loss) and produces plots of the training and validation loss curves over time.

B. Choosing Hyperparameters

1) Coarse Adjustment: Training a model on this dataset for 50 epochs takes approximately 8 hours on my machine. Therefore I chose to alter only my two most important hyperparameters in grid search: learning rate and Adam weight decay. I set the batch size to 4, which is the highest power of 2 for which my GPU would not run out of VRAM while training.

I began with a coarse grid search covering the following values:

- **Learning rate: 2e-8, 2e-7, 2e-6, 2e-5, 2e-4, 2e-3, 4e-3, 6e-3, 8e-3, 1e-2, 2e-2, 3e-2, 4e-2, 5e-2**. My learning rate values were informed by the fact that the pretrained DeepLabV3 model used a learning rate of 2e-2 [15]. Although my grid search was originally centered on this value, I later chose to extend the search into significantly lower learning rates after seeing that the lowest learning rates in my grid search tended to yield more promising loss curves.
- **Adam weight decay: 0, 1e-10, 1e-9**. I chose fewer values for weight decay given that learning rate is more important. I decided on values that follow a logarithmic scale, to adhere to common practice [16]. I kept my

weight decay values small to recognize that this task is quite nuanced and requires an expressive model.

2) Fine Adjustment: The coarse grid search resulted in the validation accuracies shown in Figure 3. Note that all the obtained Dice losses were above 0.90, which is a poor accuracy for segmentation. Despite their poor accuracy, the best of these models (the three brown rows of Figure 3 all had loss curves indicating convergence. See Figure 4 for a typical loss curve from this set. In addition, these models performed equally well on the validation dataset as the training dataset. This characteristic loss curve implied the following options to improve performance, other than abandoning DeepLabV3 in favor of a different pretrained model:

- Adding more data to the dataset
- Performing a finer grid search within the best range of the previous learning rates, in search of a better learning rate
- Running training for more epochs, to see if a further burst of improvement occurs
- Changing the optimizer to something more capable than Adam, such as Ranger, which incorporates RectifiedAdam and LookAhead and is used by one of the top Kaggle submissions [12] [6].
- Changing the label to something other than `cat`, in case the weight initialization for this label is close to a poor local minimum in the loss landscape.

I also noticed that normalizing certain images made their FTUs more difficult for the human eye to find. This effect is illustrated in Figure 5. I asked Sarah Bier, a PhD candidate in microbiology at Harvard's Mekalanos Lab, for further insight [17]. She corroborated this obfuscation effect and also recommended disabling color jitter, because the hue of the stain used encodes useful information, as different stains attach to different structures. I therefore ran a quick experiment to test whether disabling normalization and color jitter would improve the model. I trained a model for 50 epochs with the best hyperparameters from my grid search (learning rate 2e-05, weight decay 0) and disabled normalization. The result was a slight improvement in the model's validation performance, with a minimum loss of 0.9023 across all epochs. I chose to move forward with color jitter and normalization disabled.

After determining that I should disable normalization and color jitter, I took the other bulleted takeaways above into account as well. I switched my optimizer to Ranger (which we did not cover in class), increased the maximum number of epochs to 100, and ran a new grid search. However, I had to discard my old coarse grid search results, because Adam learning rates don't map well onto Ranger learning rates; I determined this experimentally after attempting to train another model and discovering that the validation loss never fell below 0.99. For my new grid search learning rates, I chose the following, based on default values and the learning rate chosen by one of the top 5 models: **3e-4, 1e-3, 3e-3**. I also changed the layer used from `cat` to `sofa` (another amorphous shape that comes in a variety of colors and patterns) and

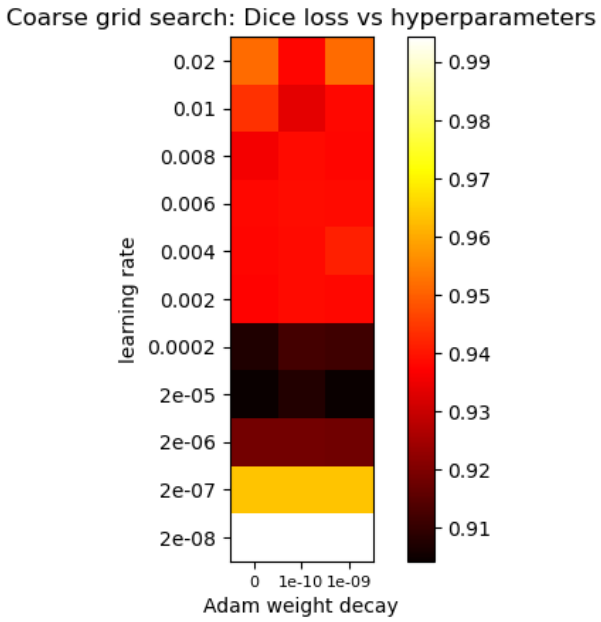

Fig. 3. Coarse grid search results. Dice loss is plotted as a function of the learning rate and weight decay for each model. For each set of hyperparameters, the dice loss shown the best validation loss of any epoch with those hyperparameters.

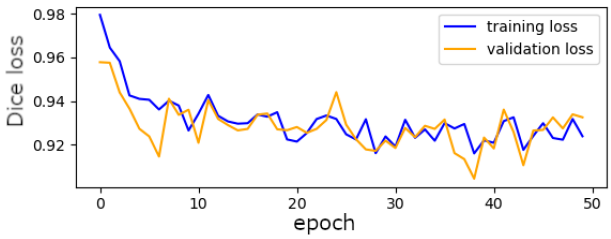

Fig. 4. Example of a model produced in coarse grid search converging. This loss curve is characteristic of all 9 models with learning rates in the range of 2e-06 through 0.0002. This particular loss curve was obtained with a learning rate of 2e-05 and a weight decay of 0.

extended my data by adding another dataset with a similar format [18].

This fine grid search yielded no improvement. The lowest validation loss across the new grid search was 0.9065, compared to 0.9043 in the old grid search. The results of the new grid search can be seen in Figure 6.

3) Sanity Check with UNet: As before with the coarse grid search, the fine grid search failed to even obtain a training loss below 0.90 – even after the adjustments discussed above. This persistent high training loss suggests a failure of the model itself to capture the nuances of the dataset. I wanted to verify whether this was true, and also verify the correctness of my code outside the choice of model. To verify these things, I tried training a UNet instead, while keeping all training parameters the same except for the learning rate, which I reduced to **1e-5** to better match the learning rates of other successful UNets [6]. I even kept the backbone the same, retaining the

Fig. 5. Normalizing certain images makes it more difficult for the human eye to find FTUs. In the original image on the left, the FTUs (shown in the center image) can be seen as dark splotches. But in the normalized image on the right, these FTUs are harder to distinguish from their surroundings.

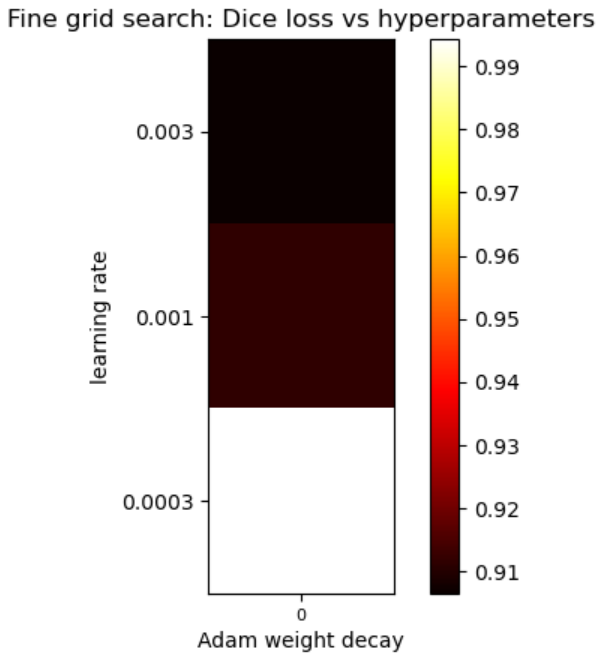

Fig. 6. Fine grid search results after adding more data to the dataset, running training for more epochs, changing the label of the pretrained label, and improving the optimizer. Dice loss is plotted as a function of the learning rate and weight decay for each model. For each set of hyperparameters, the dice loss shown the best validation loss of any epoch with those hyperparameters.

same ResNet101 architecture as with my DeepLabV3 models. Changing to a UNet immediately improved the performance, lowering the validation loss from 0.9043 to 0.1291, and improved the shape of the loss curve, as shown in Figure 7.

III. Results

I trained my models on the dataset provided with the Kaggle competition, combined with the dataset *Data for "Segmenting functional tissue units across human organs using community-driven development of generalizable machine learning algorithms"* [3] [18]. The hyperparameters of my models are shown in the final two columns of Table I. A qualitative error analysis of my models is shown in Figure 8. The qualitative error analysis shows that my best UNet model captures the FTUs almost perfectly, with the exception of the second example, where it hallucinates two extra small FTUs, misses another one, and detects another at a slightly incorrect location.

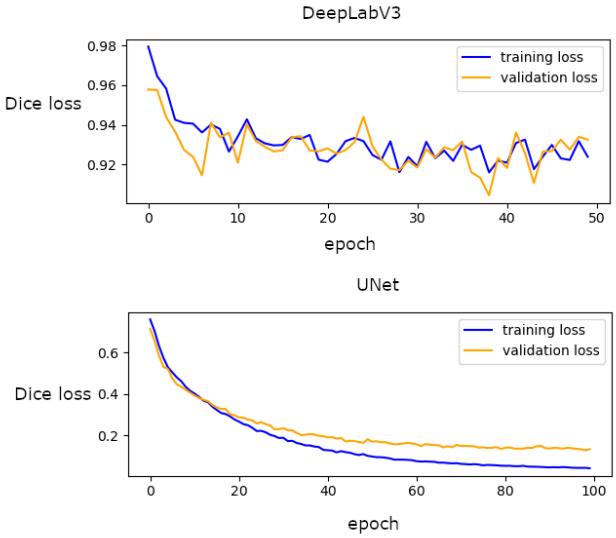

Fig. 7. Loss curves of the best DeepLabV3 and UNet models. Note that the axes use different scales and offsets. Switching to UNet with minimal changes yields a significant improvement in the Dice loss.

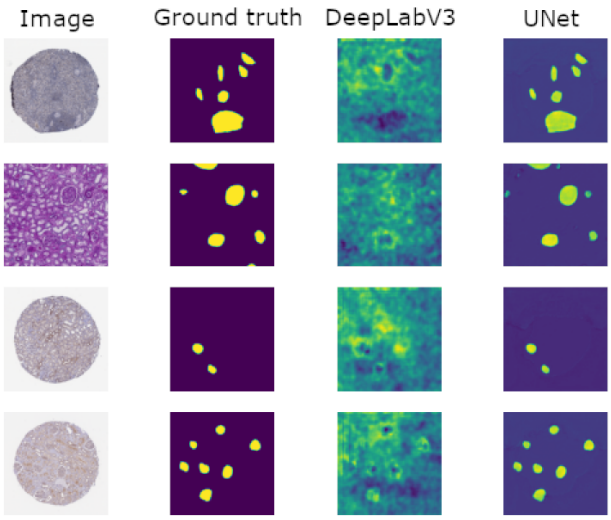

Fig. 8. Qualitative error analysis of the best DeepLabV3 and UNet models on 4 images randomly chosen from the validation dataset.

By contrast, my best DeepLabV3 model has highly diffuse logits over each image, which often become brighter at the edges of FTUs but darker inside them. Even so, its logits don't respond to all of the FTUs. For instance, DeepLabV3 makes several omissions in the second example.

The best DeepLabV3 model scores a minimum Dice loss of 0.9043, whereas the best UNet model scores 0.1291 with relatively little overfitting. Loss curves for these models are shown in Figure 7.

IV. Related Work

Godwin et al released a paper analyzing the top five submissions to the Kaggle challenge [6]. All five submissions use UNet and have Dice losses below 0.15. Table I shows a comparison of the models' design choices with my own.

Top 5 Models Comparison							
	Tom	**Gleb**	**Whats goin on**	**DeepLive.exe**	**Deepflash2**	**My DeepLabV3**	**My UNet**
Architecture	UNet	UNet	UNet	UNet	UNet	DeepLabV3	UNet
Loss	Binary cross-entropy, Lovasz	Dice	Binary cross-entropy	Cross-entropy	Dice, cross-entropy	Dice	Dice
Optimizer	SGD	Adam	Adam	Adam	Ranger	Ranger	Ranger
Max LR	1e-4 - 1e-6	1e-4 - 1e-6	1e-4	1e-3	1e-3	2e-5	1e-5
Batch size	8	8	16	32	16	4	4
Epochs	50-100	50-100	50	10,000+	Unknown	50	100

TABLE I
COMPARISON OF THE TOP 5 MODELS' DESIGNS WITH MY DESIGNS.

UNet is a popular choice among the submissions because it was developed specifically for biomedical image segmentation [5]. By contrast, I chose a DeepLabV3 model to see if its parallel runs with different atrous dilation factors would result in improved performance [4]. After experimentally discovering that DeepLabV3 isn't expressive enough for this task, I trained a UNet as well, and found that its Dice loss (0.1291) landed in the range of the top 5 submissions from before ($<$0.150).

V. SUMMARY/DISCUSSIONS/CONCLUSION

I used the Kaggle challenge *HuBMAP + HPA - Hacking the Human Body* to determine whether a DeepLabV3 model could outperform a UNet model at an FTU segmentation task [3]. I hypothesized that DeepLabV3 might be an effective model because of its robustness to scale changes, but experimentally found that it fails to capture the nuance of the task, unable to lower the validation Dice loss past approximately 0.90, even after multiple rounds of adjustments to the dataset, preprocessing pipeline, hyperparameters, and optimizer. By contrast, I experimentally found that UNet performs significantly better at this task even without adjusting the aforementioned factors except for a single-attempt adjustment of the learning rate. My UNet model achieved a Dice loss in the same range as the top 5 submissions to the original challenge [6].

The underperformance of DeepLabV3 on this task is striking, given its effectiveness on tasks outside of microscopy. A future project could investigate the effectiveness of other non-microscopy-oriented models when finetuned for microscopy tasks. Future work could also attempt to improve my UNet model further. The current UNet model was obtained simply to verify that earlier difficulties training were the fault of DeepLabV3, and not some component of my implementation – yet it performed among the top 5 submissions regardless [6].

REFERENCES

[1] Theo88, "Hubmap+hpa 1024x1024 png generation," https://web.archive.org/web/20230408183931/https://www.kaggle.com/code/theo88/hubmap-hpa-1024x1024-png-generation, accessed: 2023-04-08.

[2] S. Masubuchi, E. Watanabe, Y. Seo, S. Okazaki, T. Sasagawa, K. Watanabe, T. Taniguchi, and T. Machida, "Deep-learning-based image segmentation integrated with optical microscopy for automatically searching for two-dimensional materials," *npj 2D Materials and Applications*, vol. 4, no. 1, p. 3, 2020.

[3] Y. Jain, "Hubmap + hpa - hacking the human body," https://www.kaggle.com/competitions/hubmap-organ-segmentation/overview, accessed: 2023-04-14.

[4] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[5] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, 2015, pp. 234–241.

[6] L. L. Godwin, Y. Ju, N. Sood, Y. Jain, E. M. Quardokus, A. Bueckle, T. Longacre, A. Horning, Y. Lin, E. D. Esplin *et al.*, "Robust and generalizable segmentation of human functional tissue units," *bioRxiv*, pp. 2021–11, 2021.

[7] PyTorch, "Semantic segmentation," https://pytorch.org/vision/stable/models.html#semantic-segmentation, accessed: 2023-04-14.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[9] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[10] Z. Joudar, "Read images tiff - organ segmentation," https://www.kaggle.com/code/zakariajoudar/read-images-tiff-organ-segmentation, accessed: 2023-04-14.

[11] H. Lee, "Training with thickness and staining augmentation," https://www.kaggle.com/code/hyunwoo2/training-with-thickness-and-staining-augmentation/notebook, accessed: 2023-04-14.

[12] L. Wright, "Ranger - a synergistic optimizer." https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer, 2019.

[13] J. Y. Chan, A. P. Leung, and Y. Xie, "Efficient high-dimensional kernel k-means++ with random projection," *Applied Sciences*, vol. 11, no. 15, p. 6963, 2021.

[14] AlexisW, "Error: Expected more than 1 value per channel when training," https://discuss.pytorch.org/t/error-expected-more-than-1-value-per-channel-when-training/26274/1, accessed: 2023-04-14.

[15] PyTorch, "Semantic segmentation reference training scripts," https://github.com/pytorch/vision/tree/main/references/segmentation, accessed: 2023-04-14.

[16] Hasty, "Weight decay," https://hasty.ai/docs/mp-wiki/solvers-optimizers/weight-decay, accessed: 2023-04-14.

[17] J. Mekalanos, "Members," https://mekalanoslab.med.harvard.edu/members/, accessed: 2023-04-14.

[18] Y. Jain, "Data for "segmenting functional tissue units across human organs using community-driven development of generalizable machine learning algorithms"," https://zenodo.org/record/7545745, accessed: 2023-04-14.