Developing a Web App for the Automatic Cataloging of *Caenorhabditis elegans*

Anshul & Vaibhav Rastogi

New Hampshire Academy of Science (NHAS) Applied Scientific Research Program 2022
Mentor: Alyson Michael

Abstract -

The microscopic nematode *C. elegans* is a widely used model organism in numerous biological and toxicological studies. Assays often involve counting these nematodes manually, which is a time-consuming process.

A web app prototype is developed to catalog *C. elegans* by life stage in images of plates with worms and eggs. The app architecture depends on convolutional neural networks (CNN) and is comprised of five primary components: Input, which is an image uploaded by the user; Model 1, which is an object detection CNN to identify worms and eggs; Intermediary, which counts the eggs and crops out the worms; Model 2, an image classifier that sorts the cropped-out worms by life stage; and Output, which presents to the user the counts of worms in each life stage from the following four groups: eggs, L1-L2, L3-L4, and Adult.

Model 1 was trained for 250 epochs with the object detection framework YOLOv5 and approximately 700 images captured by an OLYMPUS OM-D E-M5 Mark III camera through a stereozoom microscope then annotated and augmented through Roboflow. Model 1 demonstrated some overfitting to training data. Model 2 was constructed with custom layers and loss functions with Tensorflow Keras. Training images were initially to be extracted as objects from the Model 1 annotations, but a Roboflow error caused the annotations to be processed incorrectly. Thus, Model 2 was trained for only 10 epochs as a proof-of-concept due to incorrect training data.

Future directions include increasing training images for both models and correcting training images for Model 2.

Keywords -

Web app; Machine learning; *Caenorhabditis Elegans*

1 Introduction

Caenorhabditis elegans (*C. elegans*) is a microscopic nematode widely used in laboratories across the globe as a model organism to conduct biological and toxicological studies [1]. They are highly useful model organisms due to their high fecundity, short lifespan, numerous evolutionarily conserved pathways, and low sustainment cost [1]. Their use is widespread and their genome and nervous system have been entirely mapped [1].

Many assays for *C. elegans* involve the manual counting and cataloging of *C. elegans* in their various life stages on nematode growth medium plates. This is a long, tedious process prone to inaccuracy.

The development of an app prototype that can automate this cataloging of the wild type *C. elegans* by life stage was started. To achieve this, CNNs (as defined in Definition 1.0.1) are relied upon.

Definition 1.0.1 (CNN)**.** A CNN (convolutional neural network) is a class of machine learning models specifically designed for the processing of image pixel data. [2]

A CNN sorts by specific features in an image (for instance: two eyes, a nose, a mouth, etc.) to make inferences about the image. Two CNN types are used for this project – an object detector, which locates particular types of objects in a given image, and an image classifier, which sorts images into classes on a single-image basis.

A visualized example can be seen in Figure 1.1.

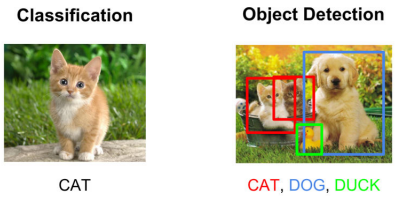

Figure 1.1: Example of image classification and object detection. Adapted from [2]

A variety of methods were used in previous attempts to automate the cataloging of *C. elegans*, as seen in works such as WormSizer [3], QuantWorm [4], and WormToolbox [5]. In Automated Wormscan

[6], for instance, *C. elegans* were identified in video feed by detecting movement. Other research, such as WorMachine [7], have used machine learning to identify *C. elegans*. However, employing machine learning to count *C. elegans* by life stage in particular is a relatively unexplored area.

The life stages of *C. elegans* are depicted in Figure 1.2.

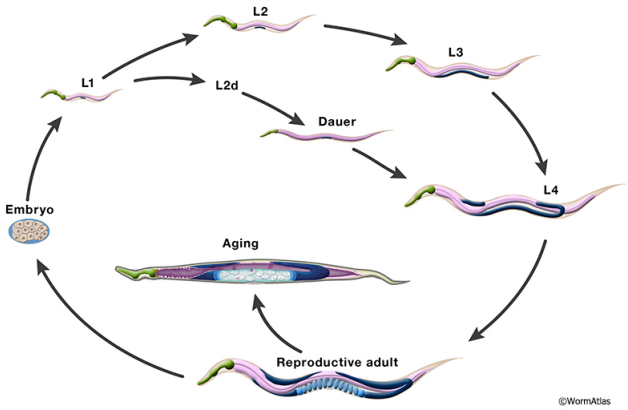

Figure 1.2: Visualization of *C. elegans* life stage as delineated by life stage. Note that "Embryo" is equivalent to a *C. elegans* egg. Adapted from [8].

2 Methods

Our app is a prototype for a browser-based application that can identify and count *C. elegans* and their eggs in a given image and then sort the *C. elegans* by life stage. The images can be uploaded to the app to receive counts of *C. elegans* in each of four life stage groups (eggs; L1-L2; L3-L4; Adult); ideally, the images can be of the quality collected from any digital camera mounted over a stereo microscope. Identifying *C. elegans* in the dauer life stage and dead *C. elegans* are beyond the scope of this project as they are rare in unstressed wild-type populations.

This section is dedicated to describing the implementation of these features in our prototype.

2.1 Architecture

The architecture of the app, as depicted in Figure 2.3, relies on two machine learning models denoted Model 1 and Model 2. Model 1 is an object detector that identifies living *C. elegans* in any life stage except dauer and *C. elegans* eggs. As this is a continuation

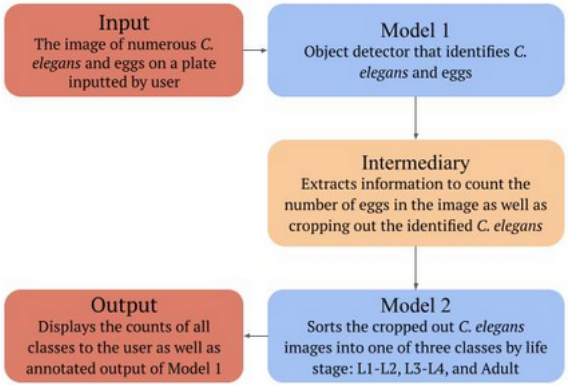

Figure 2.3: Visualization of app architecture from input (image uploaded by user) to the output (counts of *C. elegans* in each life stage class).

of work from previous years, a functioning prototype of Model 1 was previously developed with the open-source object detection framework YOLOv5; this prototype was revised with improved training data this year.

The identified *C. elegans* from Model 1 are then isolated as separate images through an Intermediary and passed forward to Model 2. Model 2 is an image classifier dedicated to sorting and counting the *C. elegans* from Model 1 by life stage. A very basic prototype of Model 2 had been constructed the prior year in Tensorflow Keras; it was expanded upon in this project.

2.2 Model 1

Model 1 was created with the PyTorch-based object detection framework YOLOv5. To train the model, images of *C. elegans* plates were first collected via an OLYMPUS OM-D E-M5 Mark III [9] camera mounted over a stereozoom microscope. In addition to the hundred or so images available from the previous year's work, approximately 900 more images were collected for a total of about 1000 images.

Although there were initially roughly 500 images available from the previous year, the majority had been acquired through splitting the frames of videos, resulting in many repeats or very similar images that caused overfitting. These repeats were removed from the dataset.

These were then uploaded to Roboflow. Of the images, 716 were annotated, including null images. A selection of such images can be seen in Figures

2.5 through 2.8. In annotated form, Roboflow highlights the hatched *C. elegans* object ("Worm") with a magenta bounding box and the eggs ("Eggs") with a lime-green bounding box (see Figure 2.4).

A variety of different image types is preferable to make the model generalizable as well as to prevent the model from making errors. All of Figures 2.5 through 2.8, for instance, contain numerous worm tracks, which are the sinuous trails left in the plate medium from the *C. elegans*' winding locomotion. As the contours of these worm tracks are similar to those of hatched *C. elegans*, it is important to include images with worm tracks to train the model in being able to distinguish between the two. Similarly, Figure 2.8 includes several bubbles that could be mistaken for eggs.

Figure 2.4: Model 1 annotation bounding box color theme. Applicable to Figures 2.5 through 2.8.

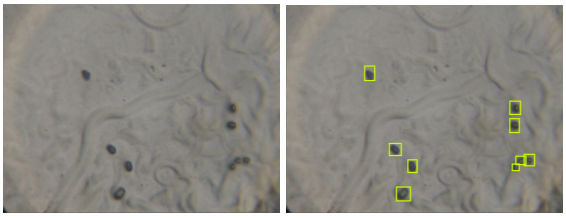

Figure 2.5: Model 1 training image of a plate with eggs before and after annotation (left and right images, respectively).

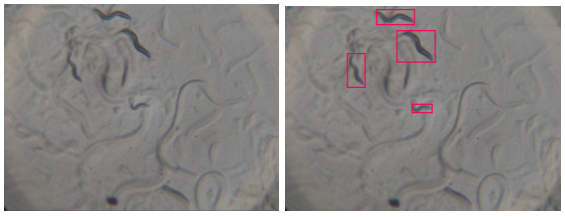

Figure 2.6: Model 1 training image of a plate with *C. elegans* before and after annotation (left and right images, respectively). Note that the black spot near the bottom of the image is debris, not an egg.

These annotated images were then exported as a dataset with the following augmentations applied:

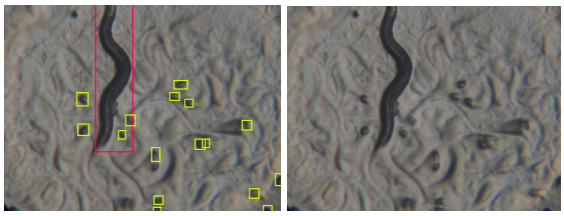

Figure 2.7: Model 1 training image of a plate with *C. elegans* and eggs before and after annotation (left and right images, respectively). Note the numerous worm tracks.

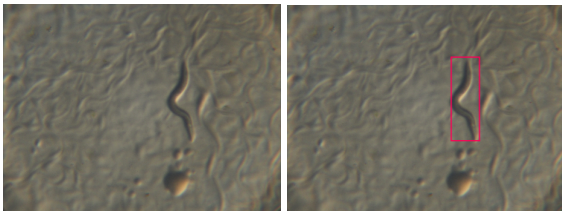

Figure 2.8: Model 1 training image of a plate with a *C. elegans* before and after annotation (left and right images, respectively). Note the numerous bubbles.

Outputs per training example: 6
Hue: Between $-25°$ and $+25°$
Saturation: Between -25% and $+25\%$
Brightness: Between -15% and $+15\%$
Exposure: Between -25% and $+25\%$
Blur: Up to $1.75px$
Noise: Up to 5% of pixels
Bounding Box: *Brightness:* Between -10% and $+10\%$
Bounding Box: *Exposure:* Between -25% and $+25\%$
Bounding Box: *Blur:* Up to 0.5px
Bounding Box: *Noise:* Up to 3% of pixels

Including augmentations, this resulted in a total of approximately 2.8k training images, 107 validation set images, and 69 testing set images. Model 1 was trained on these images for 250 epochs. See 3.1 for performance metrics.

The resulting Model 1 prototype was then exported as a PyTorch weights file (`.pt` file) for implementation into the app.

2.3 Model 2

Previously, Model 2's training had been implemented with Tensorflow Keras. As the code for this

initial version of Model 2 was lost, Tensorflow Keras was used again to construct a new version. Tensorflow Keras was selected due to the ease it allows in building sequential image classifiers with custom layers. Model 2 was set up with the following Keras layers:

```
layers.Rescaling(1/255),
layers.Conv2D(16, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(32, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(64, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Dropout(0.2),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(num_classes)
```

To train Model 2, the objects from the Model 1 annotations in Roboflow were isolated, exported, and re-uploaded to Roboflow for annotation by life stage as an image classification dataset.

However, Roboflow did not process augmentations in the reupload correctly, resulting in incorrect images; it seemed that Roboflow had misunderstood the format for bounding box coordinates (for instance, interpreting a cxcywh format as an x1y1x2y2 format), so the objects in the images were incorrectly cropped out. Nevertheless, a prototype of Model 2 was able to be tested with these images and the dataset, so the dataset was exported with the following augmentations:

Outputs per training example: 1
Flip: Horizontal, Vertical
90° Rotate: Clockwise, Counter-Clockwise, Upside Down
Rotation: Between $-45°$ and $+45°$
Shear: $\pm 15°$ Horizontal, $\pm 15°$ Vertical
Grayscale: Apply to 10% of images
Hue: Between $-25°$ and $+25°$
Saturation: Between -25% and $+25\%$
Brightness: Between -25% and $+25\%$
Exposure: Between -25% and $+25\%$
Noise: Up to 5% of pixels

This resulted in approximately 3.1k total images. Of these, 20% were taken for the validation set while 10% were taken for the testing set. For training, Model 2 was compiled with the Keras Adam optimizer, which is a variant of stochastic gradient descent and it has been demonstrated that Adam is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters" [10]. The loss function was set to sparse categorical crossentropy, which determines loss by calculating the crossentropy between classes when there is more than one class (in this case, Model 2 deals with 3 classes – L1-L2, L3-L4, and Adult).

Model 2 was then trained for 10 epochs (see accuracy metrics in 3.2). A small number of epochs were used as the training data for Model 2 consisted of the incorrectly cropped images that Roboflow produced, thus rendering any classification of images by Model 2 meaningless; instead, it was preferred to train Model 2 as a proof-of-concept and to obtain an idea of how well training proceeded with Model 2's current hyperparameter settings. Upon completion of training, Model 2 was saved in Open Neural Network Exchange (ONNX) format (`.onnx` file) for integration into the app.

2.4 App: See Elegans

Numerous APIs exist in Python to run inference sessions with PyTorch and ONNX machine learning models. Due to the ease of handling these libraries, the app prototype was constructed in Python. The working name for the app is "See Elegans."

Besides Model 1 and Model 2, several key features were identified as desirable for the app and thus implemented. These are as follows:

- A method for the user to upload an image
- A method to preview the image uploaded and run the models and the Intermediary on those images
- A method to alter the confidence threshold of Model 1
- A display of the results, including a visualization of identified bounding boxes

To construct the app UI, the Python library Streamlit was employed. This allowed for the efficient addition of buttons, sliders, and other user-interactable controls.

3 Results Discussion

Presented in this section are the metrics and results regarding the performance of the individual models.

3.1 Model 1 Results

Note that the metrics in Figures 3.9 to 3.11 are collected from the Tensorboard display of Model 1 following training.

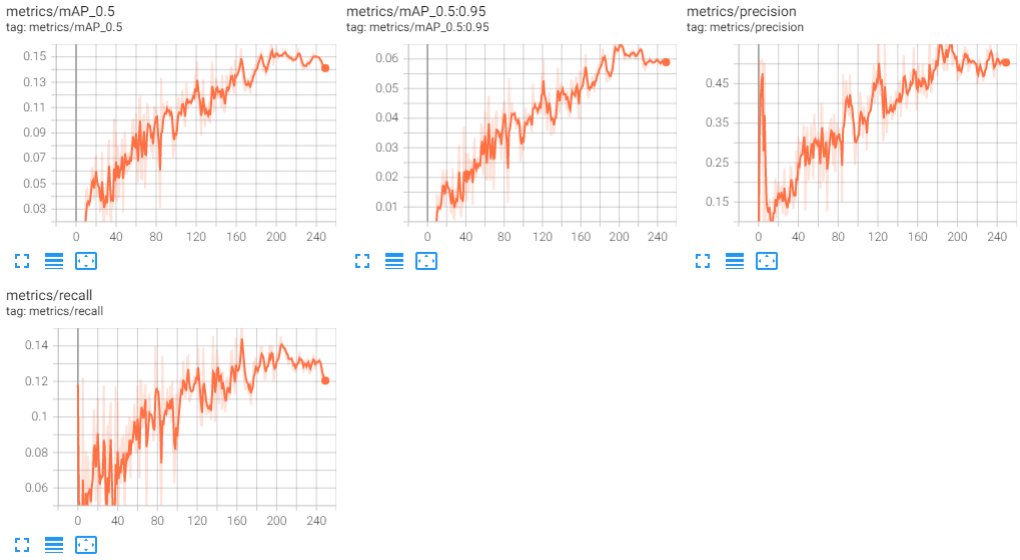

Figure 3.9: Various metrics for Model 1 over the 250 epochs of training.

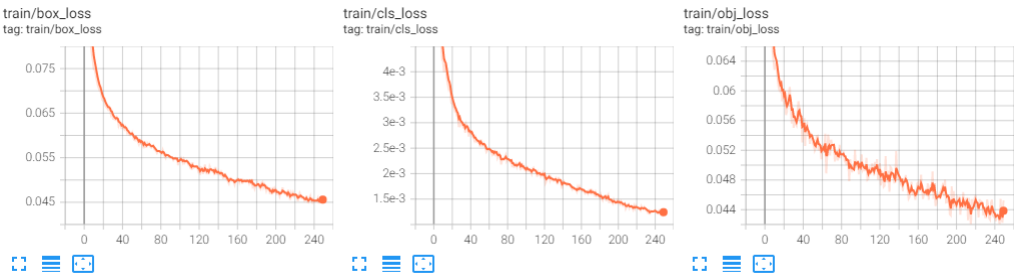

Figure 3.10: Various training dataset metrics for Model 1 over the 250 epochs of training.

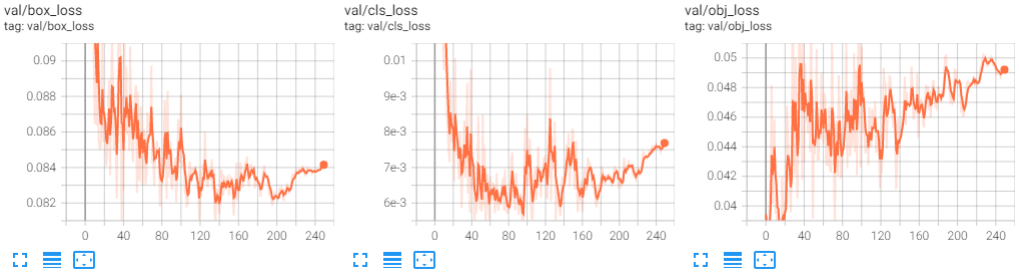

Figure 3.11: Various validation dataset metrics for Model 1 over the 250 epochs of training.

We see from Figures 3.10 and 3.11 that, although training dataset loss decreases continually over the course of training, the validation dataset box and class losses begin to trend upward after 200 epochs. This suggests that, after 200 epochs, Model 1 began to experience overfitting. This can be addressed by increasing the number of diverse images and/or decreasing the number of epochs as appropriate.

Displayed in Figure 3.13 is the Model 1's identifications of hatched C. elegans and eggs in various sample images. The bounding boxes for hatched C. elegans are blue while the bounding boxes for eggs are red.

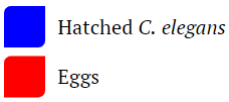

Figure 3.12: Model 1 results bounding box color scheme. Applicable to Figure 3.13

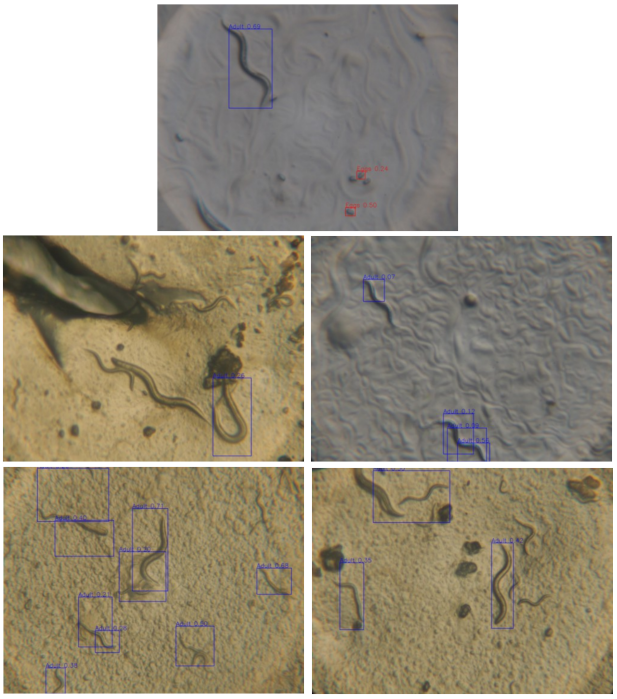

Figure 3.13: Model 1 results on sample images.

3.2 Model 2 Results

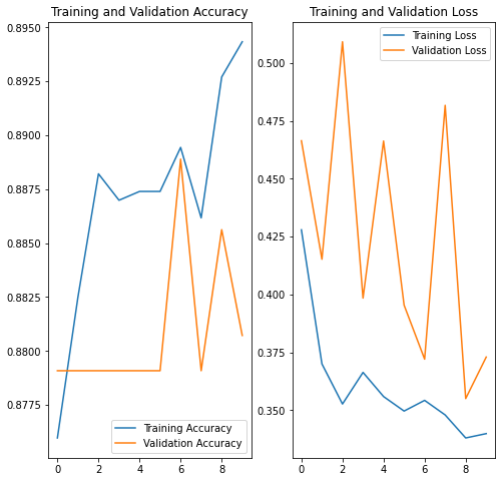

Figure 3.14: Accuracy and loss metrics for Model 2 over the 10 epochs of training.

4 Future Work

Looking forward, there are four key features to be expanded upon or added in the next iteration of the prototype:

- Improvement of Model 1 training and training data; for instance, Model 1 had difficulty identifying *C. elegans* that looped over themselves, as demonstrated in **??**. A larger, more diversified training dataset could alleviate this.
- Acquisition and annotation of correct images for Model 2; this could be achieved via manually cropping *C. elegans* from the images used to train Model 1, although such a process would be time-intensive.
- Tuning Model 2's hyperparameters for better training – that is, tuning layer types and dimensions, loss function, and other aspects of Model 2's setup that may impact training.
- The addition of a backend server that would allow the app to be browser-accessible with a permanent URL. The server could be hosted through sites such as github, ngrok, or Streamlit.

Further along the line, it may be desirable to introduce other features, such as crowdsourced image annotations or tailoring additional models for *C. elegans* strains besides the wild type. Some smaller, quality-of-life changes may also be implemented, such as memoizing the compiled models into the browser cache to decrease loading time on the site upon revisits. On a similar note, exporting Model 2 to a different file format (such as `.pt` or `.hd5`) may also be an interesting avenue as the `.onnx` format forms heavy files that are time-intensive to load.

It may also be possible to use the known error of the app's identification to analyze results in an experiment. For instance, it may be possible to determine whether or not the difference between the app's *C. elegans* counts for two different plates is statistically significant given the app's error.

5 Acknowledgements

We would like to thank our research mentor, Dr. Alyson Michael, and our primary college student assistant, Matthew Adner, for their guidance, support,

and encouragement. We would also like to thank The New Hampshire Academy of Science for this opportunity.

References

[1] Meyer BJ-editors Riddle DL, Blumenthal T. et al. *C. elegans II. 2nd edition*. Cold Spring Harbor (NY): Cold Spring Harbor Laboratory Press, 1997. URL https://www.ncbi.nlm.nih.gov/books/NBK20086/.

[2] Convolutional neural networks — ibm. URL https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[3] L. Ryan Baugh Brad T. Moore, James M. Jordan. Wormsizer: High-throughput analysis of nematode size and shape, 2013. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0057142.

[4] Celeste Riepe Weiwei Zhong Sang-Kyu Jung, Boanerges Aleman-Meza. Quantworm: A comprehensive software package for caenorhabditis elegans phenotypic assays, 2014. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0084830.

[5] Kamentsky L. Liu-Z. H. Riklin-Raviv T. Conery A. L. O'Rourke E. J. Sokolnicki K. L. Visvikis O. Ljosa V. Irazoqui J. E. Golland P. Ruvkun G. Ausubel F. M. Carpenter A. E Wählby, C. An image analysis toolbox for high-throughput c. elegans assays, 2012. URL https://pubmed.ncbi.nlm.nih.gov/22522656/.

[6] Sathyamurthy S Sukumar S-Shapira T Ebert P Puckering T, Thompson J. Automated wormscan, February 2017. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5365223/.

[7] Mor Y. Toker I.A. et al Hakim, A. Wormachine: machine learning-based phenotypic analysis tool for worms, 2018. URL https://bmcbiol.biomedcentral.com/articles/10.1186/s12915-017-0477-0.

[8] C.A.; Driscoll M. Herndon, L.A.; Wolkow and D.H Hall. Introduction to aging in *C. elegans*, 2018.

[9] E-m5 mark iii travel camera. URL https://www.getolympus.com/us/en/e-m5-mark-iii.html.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Document Template: https://www.overleaf.com/latex/templates/isarc-paper-template/cdfwbxsghpyv