Millennium Seedbank, Identifying Viable Seeds from X-Rays

— Final Report —

George Duffy, Victrix Gyasi, Aidan Holmes, Alex Morton,
Christopher Roberts, Ollie Rosen, Ivaylo Stoyanov
{george.duffy23, victrix.gyasi22, aidan.holmes23, alex.morton23,
christopher.roberts23, oliver.rosen23, ivaylo.stoyanov23}@imperial.ac.uk

Supervisors: Dr Kiran Dhanjal-Adams, Dr Rob Craven
Module: COMP70048, Imperial College London

April 29th, 2024

Contents

1 Introduction **2**
1.1 Background and Motivation . . . 2
1.2 Our Contribution . . . 2

2 Technical Decisions, Design and Architecture **2**
2.1 Dataset Creation . . . 2
2.1.1 Labelling Format . . . 2
2.1.2 Labelling Software . . . 3
2.1.3 Labelling Procedure . . . 4
2.1.4 Infestations . . . 4
2.1.5 Stratified Labelling . . . 4
2.1.6 Data Augmentation & Synthesis . . . 5
2.2 Image Segmentation Models . . . 5
2.2.1 Model Selection . . . 5
2.2.2 Mask R-CNN . . . 6
2.2.3 YOLOv8 . . . 6
2.2.4 Model Challenges . . . 6
2.3 Web Application . . . 7
2.3.1 Front-end: Design and Functionality . . . 8
2.3.2 Back-end . . . 8
2.4 Software Engineering Practices . . . 9
2.4.1 Planning . . . 9
2.4.2 Version Control & CI . . . 9

3 Model Optimisation **9**
3.1 Mask R-CNN . . . 10
3.2 YOLOv8 . . . 10

4 Evaluation **11**
4.1 Model Performance . . . 11
4.2 Application Performance . . . 11
4.3 User Survey . . . 12
4.4 Scalability . . . 13

5 Conclusion **13**

6 Future Work **14**

1 Introduction

1.1 Background and Motivation

The Millennium Seedbank at Kew Gardens acts to preserve and protect the floral biodiversity of the planet. In light of biodiversity decline due to climate change and land destruction for commercial usage, this work is more important than ever. Kew Gardens currently store over 2.4 billion seeds in their seed vault, including nearly all of the UK's natural species [Kew]. For planning re-diversification projects, it is essential to accurately record the numbers of potentially viable and non-viable seeds that are stored for future germination stages. These statistics are currently manually collected and stored in the Seedbank's database.

Researchers observe the internal structure of seeds using X-rays in order to count full, part-full, empty, and infested seeds. These seeds are classified based on the size of the endosperm and embryo relative to seed interior, and the presence of any infestations (e.g. insect larvae, fungal infections etc.). For seeds to be potentially viable they must be either filled or partially filled and not infested to any degree. This manual labelling process is time-consuming, repetitive, and prevents the Kew researchers from engaging in more interesting work.

This project's goals were two-fold: firstly to train machine learning models capable of segmenting anatomical features in seed X-rays which can then be used in downstream research, and secondly to use the segementations produced in order to automate the process of calculating adjusted seed counts (sum of full and part-full seeds).

1.2 Our Contribution

We developed a web based application that reduces the time taken to evaluate seed images and measure the number of each class, and additionally provides mask based segmentation data useful for downstream scientific tasks. A web-app was chosen as it was felt that this would be the most user friendly for a wider array of audiences. Anyone with internet access will be able to access the hosted web-app (or without access if hosted locally), whereas building the project as a Python module would have restricted the user base to those with a coding background.

The application consists of two separately deployed components: a user-facing front-end, and an inference back-end. It allows a user to select between segmentation models, interact with visualisations of the returned segmentations and seed classifications, adjust fullness thresholds for classifcation, override misclassified seeds, and download segmentation masks and adjusted seed counts. The downloadable results include additional data, such as bounding boxes, model confidence scores, and per-seed segmentation groupings.

2 Technical Decisions, Design and Architecture

2.1 Dataset Creation

A dataset of 3669 images sourced from Madagascar with an accompanying CSV file of metadata and seed classification statistics was provided by the Kew research team. Each image consisted of greyscale X-ray scans of 10-50 seeds inside of petri dishes or grids. To produce a useful training set from these required segmentation labelling of various seed components, such as seed coats, endosperms, and infestations. Figure 1 shows an illustration of some of these features.

2.1.1 Labelling Format

We selected polygon labels to annotate the various seed components since this allowed the greatest flexibility to convert between common annotation formats (such as COCO [COC] and YOLO [RF16]), ease of editing, and a straightforward mechanism to convert to pixel-based segmentation masks when required. Labelling with a polygon is as simple as clicking points around the exterior boundary of a region. Polygon labelling is faster and more precise than other methods, but the drawback is that

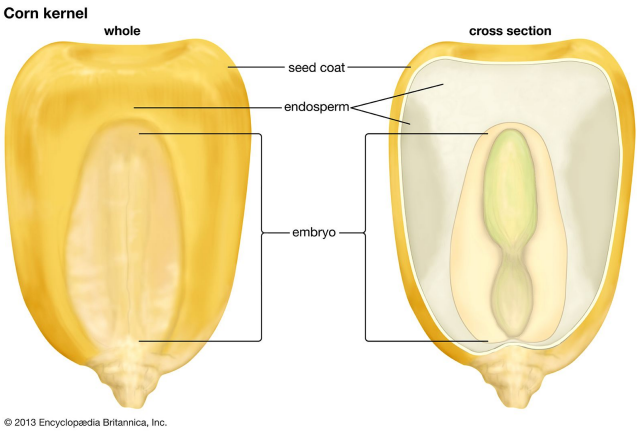


Figure 1: Anatomy of a seed. [Bri]

it does not intrinsically support creating masks that are nested inside each other. We were able to produce useful masks with a post-processing step.

2.1.2 Labelling Software

Label Studio [TMHL20], an open source labelling application, was selected for the labelling task since it allowed the group to work concurrently on labelling images on a self-hosted server, and supports a variety of exportable annotation formats. Self-hosting was deemed a key requirement for the labelling task in order to comply with Kew's restrictions on data sharing. Figure 2 shows an example of the labelling interface.

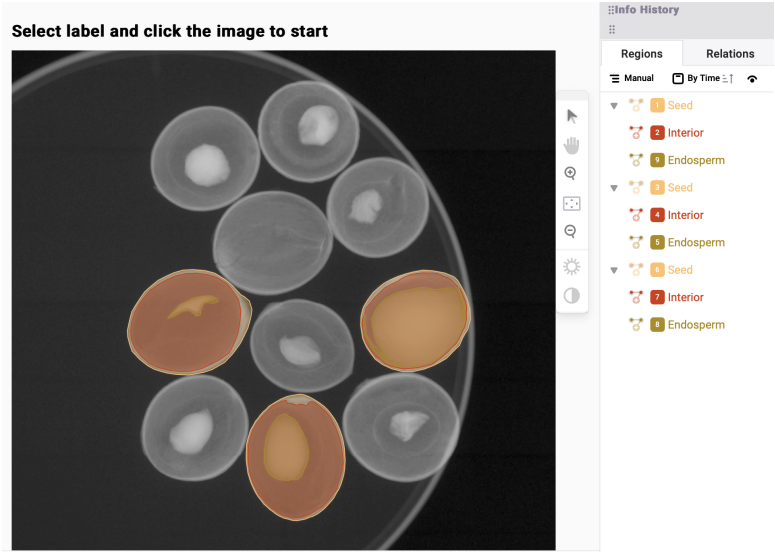


Figure 2: A partially labelled image containing 3 seeds that are labelled with 'seed', 'interior' and 'endosperm' in LabelStudio and have passed two reviews. Left: empty, middle: partially-full, right: full.

Users are able to create projects, where they define the different classes for the labels and can choose between masks and polygon labelling. The major drawback to labelling using this software is that there is no predictive labelling feature, so every label must be manually drawn. While the software is simple to use, it proved to be a very time-consuming aspect of the project. A recommendation for Kew Gardens should they wish to retrain the models with additional labelled data would be to investigate more advanced labelling software, and utilise seed data with less constraints.

2.1.3 Labelling Procedure

The key regions of a seed required to determine viability are endosperm, embryo, infestation. Polygon labelling only identifies the outer boundary of a region, so boundaries were chosen such that the outer boundary of any region nested inside another should also be the interior boundary of the outer object. We must also specify a hierarchy of region types to determine which regions can be nested within others. The following numerical hierarchy for polygon label classes was devised such that any polgyon with a lower rank is overwritten when another polygon of a higher rank overlaps with it: seed coat (1), interior (2), endosperm (3), void (4) and infestation (5). This can be seen in Figure 3.

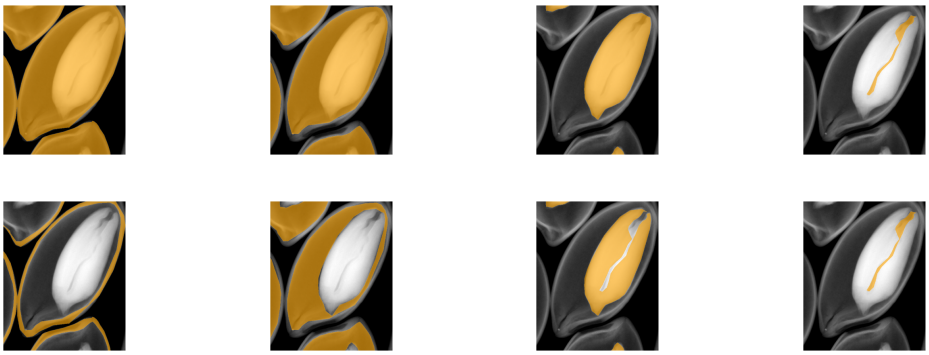


Figure 3: Example of polygon subtraction with original masks (top row) and processed masks (bottom). Notice that un-processed masks may overlap, whereas processed masks do not contain any region taken up by a higher priority region.

2.1.4 Infestations

Any seed with an infestation was deemed non-viable regardless of the size of endosperm or health of the embryo. Therefore it was very important that the models were able to identify any infestations. Unfortunately, infestations manifested in a large variety of ways in the seeds and were difficult to identify during the labelling process. Infestations usually appeared as darker areas within an otherwise uniform endosperm or pest dropping would appear as small bright pellets. A challenge arose when attempting to distinguish between genuine abnormalities caused by infestations and simply textured endosperm or voids caused by undeveloped endosperm. Often, Dr. Dhanjal-Adams was consulted to clarify such images which were then re-visited and eventually passed the review stage.

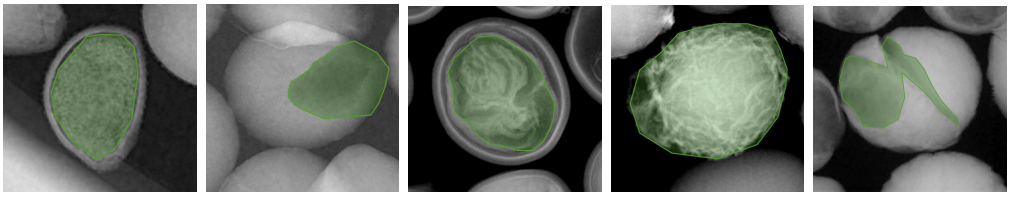


Figure 4: Examples of infestations (highlighted in green).

2.1.5 Stratified Labelling

Creating a well-labelled dataset was paramount to the success of the model training, therefore a stratified labelling structure was implemented. To ensure a well-balanced training set which encapsulated the diversity of seeds, an image from every seed family present in the dataset of 3000 X-rays available to us was selected for labelling. In the original dataset, there was an imbalance over the seed families, which meant that some families had only a single X-ray while others had up to 100 examples. Several entries in the dataset either did not have the input X-rays or corresponding classification labels. Where images were not present with accompanying data, another image from the same family was

selected. When no images with labels from a family were available, an image from a random family was selected.

Each labeller was assigned 25 images for which 3 seeds were to be labelled in each. These seeds would be used to generate the synthetic data as described in section 2.1.6. Additionally, 2 images for full seed labelling were assigned per labeller. This allowed for the generation of a fully labelled test set of 14 images for model evaluations. As part of the stratified labelling structure, every labelled image was reviewed by two other labellers to ensure consistent quality and accuracy of labels. In the first review, difficult-to-label images were tagged as "hard" from scale of "easy", "medium" and "hard". These images were reviewed by Kew researchers and seed experts. Seed annotations were checked and either passed or failed according guidelines set out in our 'Labelling Standard Operating Procedure' document. A second reviewer then checked over the labelling and gave a second opinion on the quality of labelling. Where there were discrepancies in the judgements, the images were re-reviewed with more labellers to get a conclusive decision about any inconsistencies in quality. Finally, the original labeller returned to make corrections to the failed images according to the review notes.

2.1.6 Data Augmentation & Synthesis

To accurately segment seed components the models need to be exposed to a large number and great variety of labelled examples during training. Fully labelling all the regions of a single seed image is a time-consuming process (upwards of 30 minutes per image), which rendered the labelling of even a small subset of the provided data infeasible for this project. Thus it was necessary to build a synthetic data pipeline which would allow for the generation of a large number of training examples from a set of partially labelled X-rays.

To generate a synthetic dataset of training and validation images, a synthesiser was written which consists of the following components and stages:

1. Extractor - Given a labelled source image, this extracts the image regions and annotations associated with each labelled seed
2. Transformer - Given a single labelled seed, this applies a sequence of geometric and photometric transformations (such as flipping, rotation, shearing etc.) to both image and segmentation labels to generate a new synthetic seed
3. Compositor - Given a sequence of newly synthesised seeds, this component handles the random placement of these seeds on a new image canvas, allowing for possible overlap between seeds, and generates a simulated petri dish enclosing the seeds.

Figure 5 shows a stylised example of synthesising a single image. Not only does this pipeline allow for many orders of magnitude increase in the dataset size, but it also provides the normal benefits of data augmentation (i.e. encouraging invariances to translation, scale and rotation). There is an ultimate limit to this approach since textures of different regions of the seeds are not qualitatively affected by geometric or photometric augmentations. Only a diversity of seed types in the partially labelled source images can provide this diversity.

2.2 Image Segmentation Models

2.2.1 Model Selection

To produce accurate seed segmentations it seemed natural to use a state of the art segmentation model. A survey of existing models indicated several contenders, of which YOLOv8 [JCQ23] and Mask R-CNN [HGDG17] were selected for training in parallel. This choice was motivated both out of a desire to mitigate risk (especially to avoid committing to a single model only to find out it was inadequate towards the end of the project), and also the possibility of differing performance between the models on classes of X-rays, or parts of the segmentation process. Moreover, the particular YOLOv8 implementation chosen offered additional flexibility in that retraining the model as a classifier

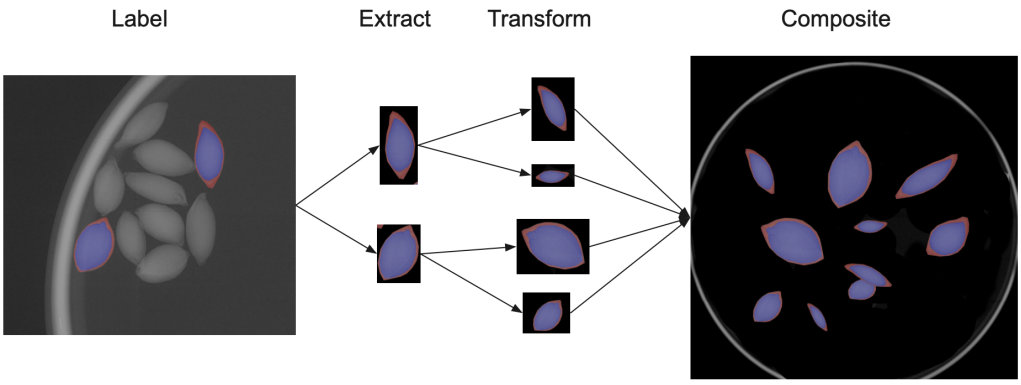


Figure 5: Data augmentation pipeline.

(rather than segmenter) would required minimal code and data changes. This kept open the option to pivot away from segmentation if it were required.

2.2.2 Mask R-CNN

Mask Regional-based Convolutional Neural Network (Mask R-CNN) is a deep learning model used for instance segmentation. It detects and outlines objects within an image, considering multiple instances of the same type of object as different. Mask R-CNN extends the existing Faster R-CNN, adding segmentation mask predictions to each region of interest in the image.

Mask R-CNN uses a backbone image classification architecture, such as ResNet50, to generate features and in parallel uses the features at each stage to generate rectangular regions of interest where the model is most confident that an object is present. This is known as the region proposal network. The image within the region of interest is reshaped into a fixed size for a final classification, bounding box prediction and segmentation. Each pixel belongs to one of a set of objects identified.

2.2.3 YOLOv8

You Only Look Once version 8 (YOLOv8) is a state-of-the-art model developed by Ultralytics which is designed to be fast, accurate and easy to use. The YOLOv8 package by Ultralytics includes several useful features - such as automatic mean average precision (mAP) evaluations during training. These scores represent the average precision computed across multiple classes of object detection in images. It serves as a comprehensive performance metric that accounts for both the accuracy and robustness of object detection algorithms. mAP was the standard evaluation metric for both YOLOv8 and Mask R-CNN. There is extensive documentation [Ult24] for using the model with Ultralytics which streamlined the implementation of training and inference for the YOLOv8 model.

2.2.4 Model Challenges

Open-source PyTorch implementations of both YOLOv8 and Mask-RCNN architectures are available online; training these models was not so easy. We faced two major issues during training: (1) segmentation models are memory intensive and (2) segmentation models are slow to train.

The forward pass when training the model produces several hundred output masks of the same height and width as the input image. The output masks, as well as all intermediates, are stored for the backward pass. A rough estimate for a single mask's memory requirement is 2000 x 2000 pixels, each a 32 bit floating point number (4 bytes), using 4 x 2000 x 2000 bytes = 16MB per output mask. If there are 50 seeds, each with a seed coat, interior and endopserm, there may be 150 masks and so 150 x 16MB = 2.4GB to store only the mask outputs for one image. Add in multiple images with all the intermediates and the memory usage grows rapidly. A forward pass with the model in training mode of 10 images typically used over 16GB of VRAM, exceeding all routinely available GPU capacities.

We verified that the memory usage was primarily due to intermediates and outputs in the forward pass by profiling memory usage throughout a training cycle, as well as using LoRA to reduce trainable parameters to 1% of the number of original model. LoRA marginally increased the total memory usage, as both the original intermediates and perturbations were stored, confirming the hypothesis that intermediates were the unavoidable cause of the large memory usage. Several solutions are available to those with large GPU memory requirements: (1) multi-GPU training (2) gradient accumulation (3) quantisation. We used gradient accumulation as the simplest of the three approaches, avoiding the complexity of multi-GPU training and the lack of support for quantisation. We used gradient accumulation with batches of 5 images and stepping gradients every 50 images, which allowed us to store the gradients without the overhead of the intermediates, reducing the memory requirements of a batch of 50 images to that of 5 images while maintaining the benefits of large batch sizes for faster convergence and shorter time per epoch.

Training a model on images comparable to the true data (similar numbers of seeds, similar image size) was slow. Mask-RCNN training took between 80 and 100 minutes per epoch, and were trained for 10 to 20 epochs. We optimised model hyper-parameters using a grid-search as grid-searches are trivially parallelisable. We ran training runs across multiple GPUs with different hyper-parameters using the available SLURM GPU cluster.

The implementation of YOLOv8 was heavily constrained by the Ultralytics Python module, which acted as a simplified wrapper for a PyTorch implementation of YOLOv8. The complexity of the module prevented us from implementing custom improvements such as LoRA and gradient accumulation. YOLOv8 training also suffered from the memory constraints, due to unpredictable spikes in memory usage causing training to halt. Training models fully took multiple days; this combined with the crashing restricted the scope of hyper-parameter search that could be performed, even when model check-pointing was used.

2.3 Web Application

The final web application is composed of a front-end responsible for user interaction, and a back-end which handles model inference. This split allows for both a separation of concerns, and flexibility in terms of the hosting of each component. The latter is particularly important for the back-end, which might need increased compute resources to handle concurrent inference requests in the future. Figure 6 shows a simplified view of a user interaction with the application.

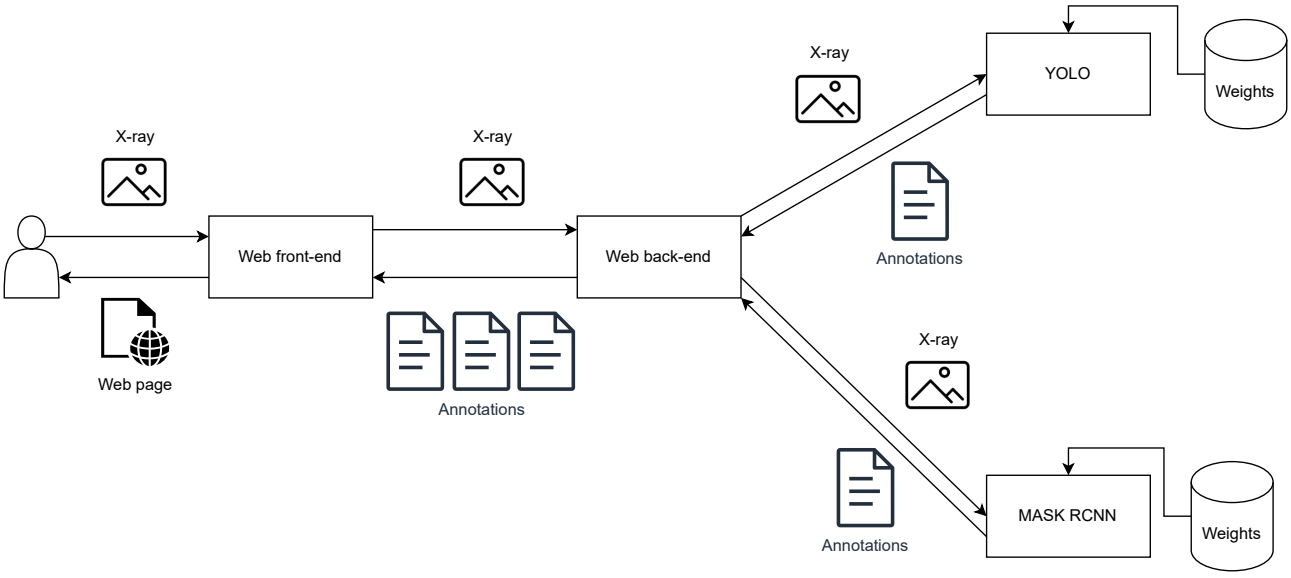


Figure 6: Application Architecture

2.3.1 Front-end: Design and Functionality

React.js was selected for the frontend due to its component-based structure and virtual DOM, which maintained modularity across the clients code-base and ensured fast rendering performance for intensive image processing tasks. Throughout the design phase, the following user story was considered:

> *"As a Millennium Seedbank staff member, I want to upload images of seed X-rays and receive an output that includes the total number of seeds, and whether each is full, part full, empty or infested. This will help me determine potential seed viability, and estimate adjusted seed count. I'd also like to easily see seed segmentations so I can use them for other scientific research."*

Given these priorities, we built the web application to: (1) upload an X-ray, (2) overlay segmentation masks on individual seeds, (3) return seed classifications, and (4) allow the users to retrieve this information in a downloadable format.

When the user starts a session, they can upload X-rays in PNG or JPEG format. After selecting a segmentation model, an inference request is dispatched to the back-end. Once the response is returned, the web client infers seed classifications based on the segmentation results. The user can then overlay classification masks on top of the original X-Ray to easily identify seed classifications, and can dynamically overlay segmentation masks on cropped images of individual seeds. We designed the front-end to include mask overlay so the user – as defined in their requirements above – can easily determine potential seed viability across both a batch of seeds and easily see segmentations for individual seeds. Due to different morphologies across seed species, different levels of endosperm coverage constitute different levels of 'fullness'. For this reason, the application also provides thresholding functionality to give the user control over the classification algorithm. Additionally, the user has overwrite functionality to replace the inferred classification returned by the algorithm as a safety net. Finally, we return seed fullness, endosperm intensity, and a histogram of seed classifications for the user to review. Importantly, none of this post-inference interaction requires additional inference requests, so interaction can occur in real time. Finally, the classifications and segmentation results, can be downloaded in two formats: a CSV (.csv) file to download the classification results or a ZIP (.zip) file to download the segmentation results. The CSV contains one row of data per X-ray image, including its image filename, the total number of seeds detected, and the number of seeds across the full, part-full, empty and infested classifications. The ZIP contains information about individual masks associated with that particular X-ray image, such as the seed ID, the mask label, its area, mean intensity, coordinates (bounding box), a base64 encoded representation of the mask png, and the models confidence.

In terms of design principles, we prioritised simplicity and control so Seedbank staff members can immediately use the application without a user guide. When presenting our interface design to Kew Gardens' staff on an external trip, we received positive feedback about its usability.

2.3.2 Back-end

The back-end comprises a Python Flask application, with a single 'predict' endpoint to handle requests from the front-end. An incoming request contains the uploaded image and key for the requested model. The back-end dispatches this image to the requested model and collects the segmentation results into a standardised model annotation object. An additional step of post-processing is then required to allocate annotations to their respective seed entities (since neither model predicts annotation hierarchies). This final object is serialised and returned in a response to the front-end.

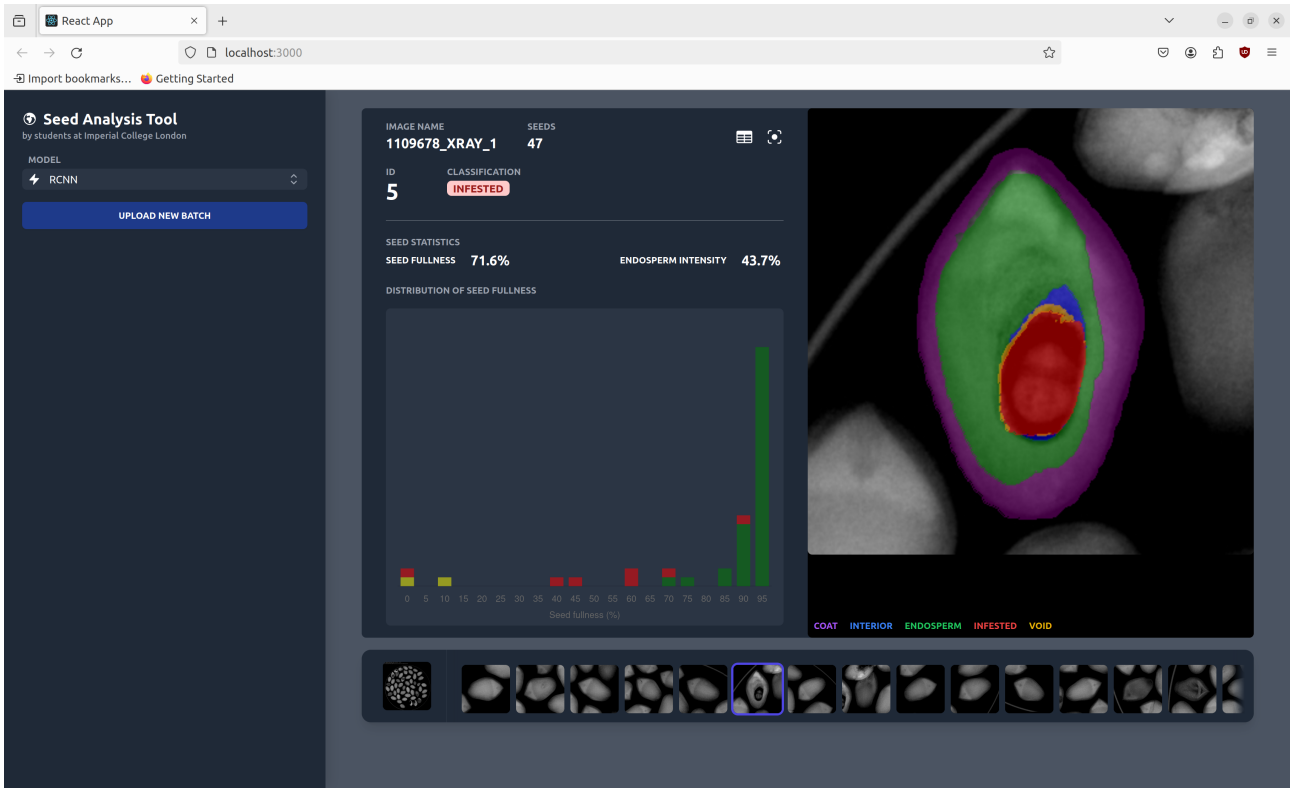


Figure 7: User-facing webpage

2.4 Software Engineering Practices

2.4.1 Planning

Work was planned at bi-weekly team meetings, focusing on the fastest path to getting the minimum viable product. We divided work according to the skills of individuals in the team, as well as using the opportunity for learning this project provided to allow people to work on skills they wished to develop. We managed the project using a combination of Trello for task management and Notion for project overview, where tasks and time frames were assigned to individuals. This also allowed updating the other team members about progress with their tasks. Slack was also used for communication and new channels would be created for relevant tasks with the relevant people. Bi-weekly meetings were conducted internally to plan our work, with fortnightly meetings with Kew Gardens to update them on our work and get feedback.

2.4.2 Version Control & CI

The team used a git repository (hosted on the department's GitLab instance) for version control. A feature branching workflow was employed, with reviewed merge requests to introduce changes to the master branch. Unit test and code linting pipelines were configured to run automatically on every push and merge-request respectively. This ensured that code quality standards were maintained, and reduced the likelihood of introducing bugs during code changes.

3 Model Optimisation

We optimised our model hyper-parameters with a grid search. We chose the grid search strategy over alternatives like random search or Bayesian Optimisation because grid searches are easily parallelisable, meaning we could run multiple independent training runs on different GPUs with little additional effort. In addition to being comprehensive and parallelisable, grid searches are deterministic and therefore reproducible.

3.1 Mask R-CNN

We conducted a two-stage grid-search for the Mask-RCNN hyper-parameters, with results presented in Table 1. We divided the search space into two groups, one which impacts gradient based steps or model weights, and another which impacts evaluation and post processing. This allowed us to train fewer models, and search over the hyper-parameters that are independent of training in a time-efficient way. We further divided training related hyper-parameters into sub-groups which we expected not to interact strongly, and optimised each sub-group independently.

Hyper-parameter	**Grid-Search Group**	**Search Values**	**Best Values**
Number of Epochs	1	10, 20	20
Learning Rate	1	1e-3, 1e-4, 1e-5	1e-4
Batch Size	1	50, 100	50
Target Mask Subtraction	2	True, False	True
Maximum Detections	3	100, 200	200
Output Mask Subtraction	Non-training	True, False	False
Binary Threshold	Non-training	0.1, 0.25, 0.50, 0.75, 0.9	0.5

Table 1: Mask R-CNN hyper-parameter search space.

The configuration using best values in Table 1 was deployed in the web-app. Should Kew Gardens wish to train new Mask R-CNN models, we encourage a similar hyper-parameter tuning project over the group 1, 3 and non-training hyper-parameters, as it is possible that the optimal values for these may change. Target mask subtraction (group 2) should not be altered from its stated optimal value as this has a detrimental impact on performance.

3.2 YOLOv8

A similar hyper-parameter grid search for the YOLOv8 model was conducted over the hyper-parameter space presented in Table 2. The training of the YOLOv8 model was highly memory intense, with even small batch sizes of 1-3 at a time resulting in the use of over 48GB of GPU RAM. It was hard to predict when the memory requirements would spike, and as the largest single GPU available to us had 48GB of GPU VRAM, the YOLOv8 grid search would run out of memory before completion. The grid search was extremely expensive in regard to time, as such a low batch size was needed. Nevertheless, we were able to work through a reasonable section of the hyper-parameter space using model checkpointing to restart a training run from the point that it crashed.

Hyper-parameter	**Explanation**	**Best Values**
Number of Epochs	Number of iterations over the dataset	200
Initial Learning Rate	Initial learning rate to allow for LR scheduling	0.01
Final Learning Rate	Final learning rate to allow for LR scheduling	0.01
Momentum	Rate to smooth gradient descent	0.937
Weight Decay	Regularization to reduce complexity	0.0005
Patience	Patience before early stopping	100

Table 2: YOLOv8 hyper-parameter search space

The configuration with best found values within the explored subset of the YOLOv8 hyper-parameter space, which is presented in Table 2 is what is applied within the web-app.

4 Evaluation

4.1 Model Performance

We compare the mean-averaged-precision against held-out test data on fully annotated images for both models, as well as on an augmented synthetic test dataset using unseen seeds. The performance of the models on each dataset is presented in Table 3

Dataset	YOLOv8	Mask R-CNN
Held-out Dataset	0.15	0.26
Synthetic Dataset	0.23	0.19

Table 3: Macro-averaged mAP scores of YOLOv8 and Mask R-CNN on different testing sets.

Each model performs differently across the two different testing sets. State of the art best performing models on the full COCO dataset achieve a mAP score of up to 0.45 on the test set. While our best performing model achieves a mAP of 0.26, we note that our dataset size is substantially smaller (∼300,000 original images for COCO vs ∼150 for our dataset). Segmenting seeds involves predicting nested masks on X-ray images, qualitatively changing the type of task from the one the models were designed for (non-nested masks on image data), so the score of 0.26 mAP represents reasonable performance. Crucially, Mask R-CNN performs better on the held-out set of true seed data as it can differentiate objects of similar textures (a petri dish vs seed coat) more easily than YOLO. We hypothesise that the two stage architecture of Mask R-CNN allows it to first locate seeds and then segment them, while YOLO performs segmentation and localisation of seeds in a single pass. As such, Mask RCNN was set to the default option of the web app. Nonetheless, YOLOv8's improved performance on the synthetic set indicates that YOLOv8 performs better on more standard images with fewer infestations, validating our decision to include it as an option on the web app.

4.2 Application Performance

To measure the performance of the application, inference request times and memory usage were profiled. A sample of 100 images was drawn at random from the provided dataset, and total inference time and peak memory usage were recorded for each image. If we consider the 95th percentiles of these distributions we can say that most requests will complete within 11.5 seconds (13.6 seconds)[1] and consume no more than 8.0 GB (6.3 GB) when using Mask R-CNN (YOLOv8).

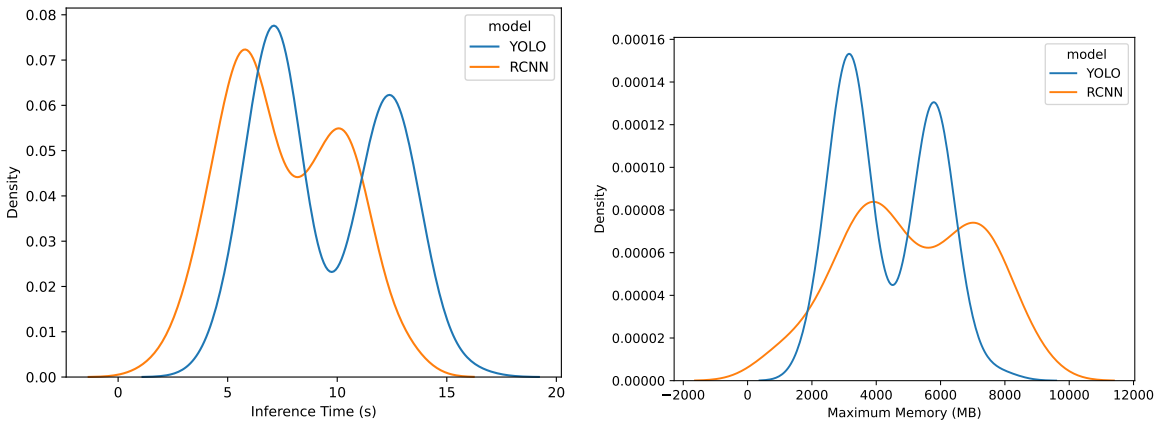
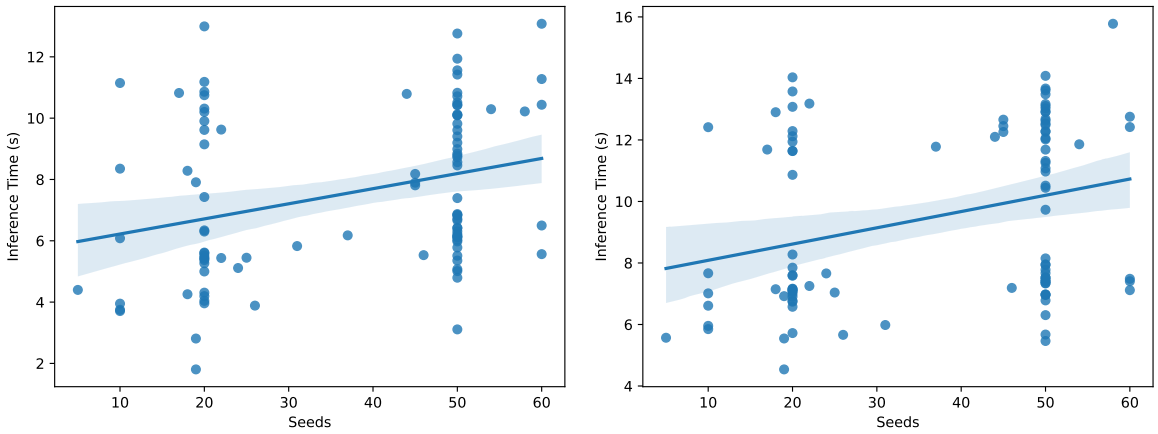


Figure 8: Inference runtime and peak memory usage

Figure 8 shows the distribution of run times and peak memory usage for inference requests on the sampled images. Both show pronounced bimodality, which arguably reflects the distribution of seeds in the sample (most X-rays contain either 20 or 50 seeds).

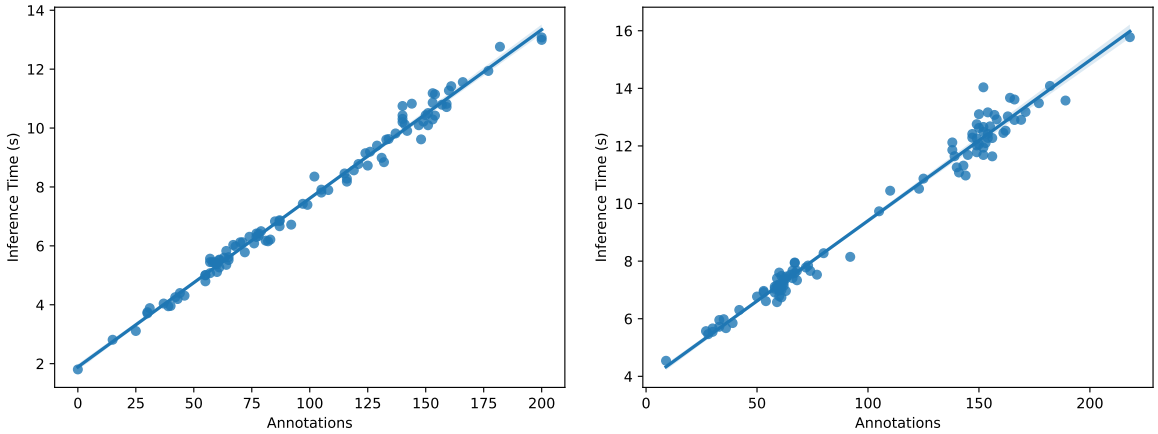
[1] When run on an Intel i7-13700H processor

Figure 9 illustrates the relationship between the number of seeds in an image, and inference runtime. Though there appears to be a weakly positive relationship between the two, there is substantial variance in runtimes both within images with comparable seed counts and between images with differing seed counts. We attribute this to the number of detectable features (seed coats, endosperms, infestations) which can vary between images with the same number of seeds. Figure 10 shows runtime as a function of the number of segmentations in an image, which exhibits a much stronger linear relationship in both models.



(a) RCNN (b) YOLO

Figure 9: Inference runtime as a function of the number of seeds in an image



(a) RCNN (b) YOLO

Figure 10: Inference runtime as a function of the number of annotations

4.3 User Survey

A product demonstration was given to Dr. Dhanjal-Adams after completion. She was then given the opportunity to use the product herself end-to-end. Finally, a survey was conducted which consisted of four quantitative questions, answered using a scale of 1 (very poor) to 10 (very good). The quantitative results of the user response are summarised in Table 4.

Dr. Dhanjal-Adams gave 10/10 ratings for clarity of the interface and downloadable results as seen in questions 1 and 4 in Table 4. She elaborated that the interface was *"very user friendly, easy to understand and use. I think I would be able to use it, even without a tutorial."* and the downloadable files had *"very clear labels, easy to understand."*

Regarding the time taken for inference, she responded that *" [It] runs within 20s which is very manageable. Very nice that it is now running locally and the benefit of running locally (rather than on*

Question	User Response
1. How straightforward is the interface to use?	10/10
2. Is model inference performed in a timely manner?	9/10
3. How accurate are the segmentation predictions?	7/10
4. Are the files in the downloadable CSV and ZIP files easily interpretable?	10/10

Table 4: Quantitative user responses to a survery after using the final product.

a server) far outweights any small inconvenience from slower processing time."

In terms of the accuracy of predictions, she awarded a lower score of 7/10 with the following justification: *"The segmentations that are done are generally correct, although some seeds are missed out and some areas are misclassified (10%), and some classes are more accurate than others. This has been discussed together, and having a manual option for relabelling the classifications is a nice approach, as are the manual filtering options (% full). Being able to download the segmentations also means we can create a dataset we can tweak and retrain the model with in the long term - very nice."*

Dr Dhanjal-Adams concluded with the following statement: *"Really nice and user friendly app, very intuitive to use. I really appreciate the option to have it run locally as this creates a lot of flexibility for users. I also appreciate the option to relabel the classifications within the app before downloading and the option to click through individual seeds and view the classifications. Similarly, the option to download the segmentations is very useful as it will allow us to use this data for other scientific applications, but also to tweak the segmentations as needed with expert input, which in turn will allow us to retrain the system in the longer term. Very nice app!"*

4.4 Scalability

The synthesised data set was created using 158 labelled images, out of a total provided 3669 unlabelled images. Significant effort was made to include images from as many species as possible, but image labelling was time-consuming. While we had the support of Millennium Seedbank researchers' advice, labelling was done by students and as such was likely lower quality than would be expected of a researcher. The ability of our models is a result of the quality and quantity of training data. Figure 11 shows how the performance scales with the size of the dataset as a direct proxy for number of source images. As the size of the dataset was increased and more source images included in making the synthetic dataset, the generalisation performance to unseen seed species increased almost linearly. The gradient of the increase suggests that with 500 to 1000 labelled source images (instead of our 158), the performance of our models on the segmentation task could increase to a level similar to the state of the art on other tasks.

Time and memory complexity are key considerations for scalability. Incorporating new data would improve performance at the cost of increased time and memory to train. Training with 158 source images took 30 hours, and the time associated with training scales linearly with dataset size, so the time to train a model on 500 source images may take up to 100 hours. In practice, one could use the increased number of source images to increase diversity in the synthesised dataset, keeping the total dataset size fixed in order to maintain similar training times. While the work achieved in this project can be scaled, Kew Gardens would need to plan retraining with due consideration for the costs associated.

5 Conclusion

A web-app was developed for the labelling and segmentation of seed X-ray images for use by Kew Gardens researchers. This was done to reduce the amount of labour-intensive labelling work required of the researchers, freeing up their time for more interesting research. Two state-of-the-art models, YOLOv8 and Mask R-CNN, were fine-tuned on synthetic seed images. These synthetic images were

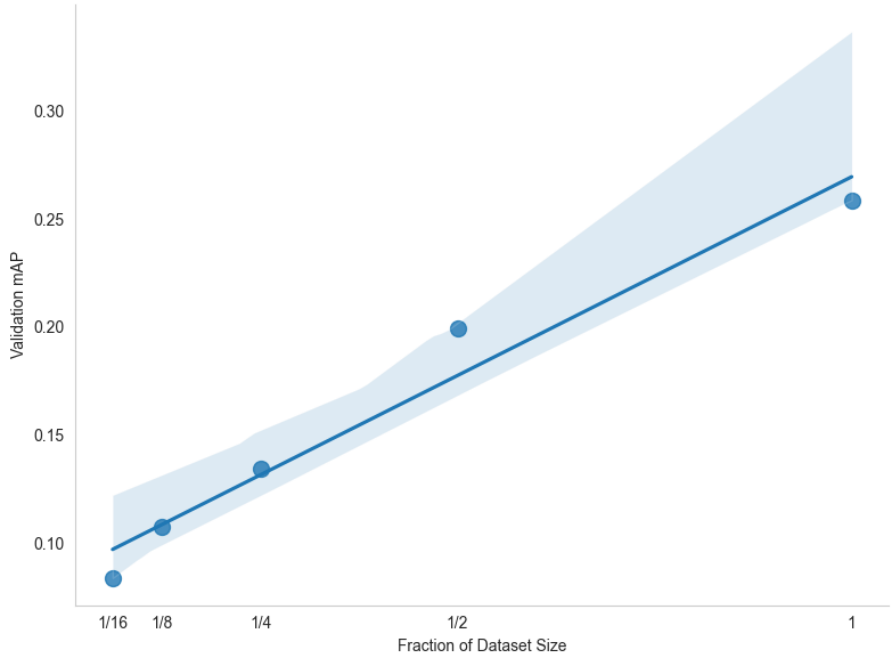


Figure 11: Maximum Validation mAP score vs Fraction of Synthesised Dataset (with number of source images in proportion to dataset size). A linear relationship is shown, implying that further data would further improve mAP score.

generated through the augmentation of real labelled images, allowing for a large labelled dataset to be generated efficiently. The web-app allowed the users to select which model they wished to use and compare results, as it was found that model the better performing model may vary for specific seed families. A user survey revealed that researchers found the web-app easy-to-use, clear and able to return results in a timely manner. While still receiving a positive response, feedback from the researchers indicated that the segmentation predictions could benefit from being more accurate. We highlight the lack of high-quality labelled data provided to us, and the fact that we lacked the time and domain specific training to generate a large labelled dataset ourselves. During the development of the models it was observed that their performance improved as the seed dataset size increased. We believe that model performance can improve further with increased quantity and quality of original labelled seed data.

6 Future Work

As part of the development process, multiple design additions were noted for potential future improvements to the web-app, and the technology that powers it.

This project aimed to achieve two goals: (1) the segmentation of seed images for scientific use; (2) the classification of seed viability. We achieved (2) using image segmentation followed by rule based classification of seeds, but alternative methods such as direct classification using methods such as bounding box classification could be simpler and more accurate. Directly labelling whole seeds as “full”, “infested”, etc. would require re-labelling of the data, and would reduce the efficacy of augmentation, but labelling the data would be faster as it requires labelling whole seeds with a single bounding box and classification instead of multiple polygons, one per seed part. A separate model built for direct classification may prove more accurate than the current joint segmentation/classification system as the data would be of higher quality and directly pertains to the overall task. This model would have a lower memory usage and shorter response time, which would be useful when researchers only require the counts of “infested”, “empty” and “ok” seeds, as is typical during day to day operations at the seed bank. The training time would also be shorter, allowing faster adjustments to improve performance as well as lower training costs. The slower and more memory intensive segmentation models could be

used only when seed segmentation is necessary.

We identified our data labelling quality as a key limitation to the accuracy of our models. Hosting a label-studio server, or a similar labelling software, on the web-app would allow researchers to provide accurate seed labels across a larger array of seed species. The researchers would be able to improve models by retraining with new data. Our web application already allows seed classifications to be altered, providing additional high quality data with which to train bounding box classification models.

If there was more time available for the project, it would be beneficial to develop and host a database that would be continuously updated as users conduct inference on their uploaded images. This would allow researchers to access current information about the state of the seed batches and their health. Once germination tests are complete on a certain batch, users could revisit the database and include new information about germination success. If persisted, this would allow researchers to check whether the predictions about potential viability were true and this may help them make more accurate predictions in the future.

References

[Bri] The Editors of Encyclopaedia. Britannica. Endosperm. https://www.britannica.com/science/endosperm. Accessed: 2024-04-15.

[COC] COCO Consortium. COCO – Common Objects in Context. https://cocodataset.org/.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[JCQ23] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8. https://github.com/ultralytics/ultralytics, 2023.

[Kew] Kew, Royal Botanical Gardens. Millennium seed bank. https://www.kew.org/wakehurst/whats-at-wakehurst/millennium-seed-bank. Accessed: 2024-04-28.

[RF16] Joseph Redmon and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[TMHL20] Maxim Tkachenko, Mikhail Malyuk, Andrey Holmanyuk, and Nikolai Liubimov. Label Studio: Data labeling software. https://github.com/heartexlabs/label-studio, 2020.

[Ult24] Ultralytics Inc. Ultralytics yolov8 documentation. https://docs.ultralytics.com/, 2024. Accessed: 2024-04-28.