Using Timelapse Imagery to Describe High-Resolution Phenology Patterns in *Silene acaulis*

Asger Svenning | au644314

January 20, 2023

Project type: Biologisk Projektarbejde (10 ECTS)
Main Supervisor: Phillip Francis Thomsen (pfthomsen@bio.au.dk)
Co-Supervisors: Toke Thomas Høye (tth@ecos.au.dk) & Hjalte Mads Rosenstand Mann (mann@ecos.au.dk)

Table of Contents

Abstract

Silene acaulis is a long-lived arctic cushion plant, which can produce a large amount of flowers per individual. In this project I show how using timelapse imagery and deep learning it is possible to conduct ecological analyses of the flowering and fruiting phenology of *S. acaulis*.

Using this approach I show how fruit setting success is influenced by the flowering length and temperature of individual flowers. I also provide a mechanistic hypothesis for how the within-cushion flowering variability is affected by phenotypic plasticity in response to cold stress, while I am not able to confirm the hypothesis due to the limited number of *S. acaulis* cushions analyzed in this project, the results show a tendency in favor of greater within-cushion flowering phenology variability in colder environments.

This project serves as an example of how the fast-growing field of deep learning can be integrated in biological research in order to simultaneously increase the scale and detail of data collection, while decreasing the manual workload associated with scaling biological field data collection.

1 Introduction

Silene acaulis, or Moss Campion, is a cold-tolerant flowering plant native to Arctic and Alpine regions in the Northern Hemisphere. *Silene acaulis* is characterized by a cushion-like morphology with the ability to produce a large number of pink-to-white solitary flowers, typically in the range of tens to hundreds per cushion. Previous efforts to describe the ecology of *S. acaulis* have been limited to genetic or cushion-level analyses, not least due to the challenge of tracking the temporal dynamics of the large number of flowers at a larger scale (Gehring and Delph 1999; Peterson et al. 2018; Waddle 2017).

At the same time the use of semi or fully automated data collection in ecology from either ground-based or satellite images (remote-sensing) has gained increased traction in recent years (Christin et al. 2019; Ferreira et al. 2020; Mann 2022; Norouzzadeh, Morris, et al. 2021; Norouzzadeh, Nguyen, et al. 2018). As such deep learning represents a promising approach for describing the phenology of *S. acaulis*.

The goal of this project is twofold: (1) to investigate differences in the flowering phenology of *S. acaulis* across different habitats and their impact on fruit setting success, and (2) to demonstrate how modern deep learning algorithms can be used to extend the scope of biological studies, even when faced with tasks that may seem insurmountable, such as manually tracking hundreds of flowers on hundreds of cushions over long time-frames using timelapse images.

The hypothesized mechanistic driver behind the first phenological pattern in *S. acaulis* is a link between within-cushion temporal flowering distribution and phenotypic plasticity in the cushion concavity in response to temperature. This hypothesis is based on previous research on *S. acaulis* which has shown that the concavity of *S. acaulis* cushions is determined by a cold-stress response (Dietrich and Körner 2014). By increasing the cushion density and concavity *S. acaulis* is able to decrease surface-to-volume ratio and increase insulation, which allows *S. acaulis* to extend it's

growth period. Thus, *S. acaulis* cushions in colder habitats are typically more concave than those in warmer habitats (Dietrich and Körner 2014). This has the side effect of creating a more pronounced light vs. shade side on the cushions of plants in cold habitats.

Specifically I hypothesize that *S. acaulis* plants in cold habitats with a more pronounced light vs. shade side will display a wavelike spatiotemporal flowering pattern, starting on the light side and moving across the top of the cushion towards the shade side, resulting in a relatively longer flowering season in colder habitats. On the other hand, the flatter *S. acaulis* cushions in warmer habitats are expected to flower in unison.

Furthermore, previous work on other arctic-alpine plant species such as *Dryas sp.* have shown that both snowmelt and flowering timing and temperature plays a central role in the phenology and flowering abundance (CaraDonna et al. 2014; Post et al. 2008; Wheeler et al. 2015). Previous works have put much focus on the effect of temperature shifts under global climate change, showing that lower flowering temperatures are associated with a greater probability of frost-damage, which could lead to unsuccessful flowering (CaraDonna et al. 2014; Post et al. 2008; Wheeler et al. 2015). However, due to the limited number of study years it is only possible to incorporate the observed temperatures, while temperature shifts are unable to be covered. Instead, I will perform an exploratory analysis on the flowering patterns which may distinguish succesful and unsuccesful fruit setting. Based on the previous research I expect that higher flowering temperatures will lead to higher fruit setting success probabilities, while lower flowering temperatures result in higher variance in the probability of fruit setting.

As with many contemporary biological studies, the data collection process is the foundation which analyses build upon. In this project the data collection process consisted of two separate steps, the first was done by Rebekka Ween in her Masters Thesis (Ween et al. 2022) and consisted of manual annotation of more than 800 000 *S. acaulis* flowers/fruits[1] in various life-stages, while the second is a multistage automated annotation and tracking procedure which builds on work by myself and Simon Sataa-Yu Larsen (Svenning and Larsen 2022).

The previous works provided the backbone of this project in the form of training data (Ween et al. 2022) and a rigorously trained model (Svenning and Larsen 2022) for predicting flower/fruit bounding boxes. This project utilized the trained model for predicting bounding boxes on a much larger set of timelapse-images of *S. acaulis*, and applying a post-hoc tracking algorithm (Zhang et al. 2021) to connect the bounding boxes between images in the timelapse.

[1]The vast majority of these annotation were withered flowers, which were disregarded in this project.

2 Methods & Materials

In this project I showcase how, through the use of deep learning it is now possible to extract valuable information from a large dataset of timelapse images, through the use of a "small" hand-annotated subset of images. Such tasks have now become common across the natural sciences in research on a wide array of topics such as star/galaxy classification in cosmology (Baqui et al. 2021; Kennamer et al. 2018) as well as in the industry for uses such as self-driving (Gupta et al. 2021; Simhambhatla et al. 2019) and agricultural pest-monitoring (Burhan et al. 2020; Liu et al. 2019; Sun et al. 2018).

Here I utilized a large dataset of images collected by Toke T. Høje and team using timelapse cameras across four study locations in Greenland and Svalbard (Narsaasauq, Bjørndalen, Ella Island and Kobb Island; see Figure 1) over multiple years (2019-2021). The full size of this dataset amounts to about 10 million images, however the hand-annotated dataset used to train a deep object detection model, `YOLOFlower`, capable of extracting the information relevant for this project (flower/fruit detection for *Silene acaulis*) consists of "only" about 2400 images (Svenning and Larsen 2022; Ween et al. 2022). Links to all scripts can be found in the supplementary materials.

2.1 Flower Detection Model

In Svenning and Larsen (2022) we opted to use `YOLOv5` (Jocher et al. 2022), an extremely popular semi-opensource model which is relatively easy to work with, as the backbone for the deep learning model used in this project dubbed `YOLOFlower`. Previous work on adapting existing frameworks to this kind of data have been hampered by the small sizes of the objects (Toke T. Høje & Hjalte Mann, personal correspondence), which we confirmed and solved by using sliced training and inference (Svenning and Larsen 2022) by building upon the `SAHI` framework (Akyon et al. 2022). Another major issue which is not unique to any specific machine-learning framework is that biased data results in biased models. In the case of this project the proportions of buds, flowers, immature and mature fruit in *S. acaulis* cushions are inherently massively skewed as most flowers do not lead to successful fruiting. This problem was solved by the use of Inverse Class Frequency Weighted Loss in the training of `YOLOFlower`, which

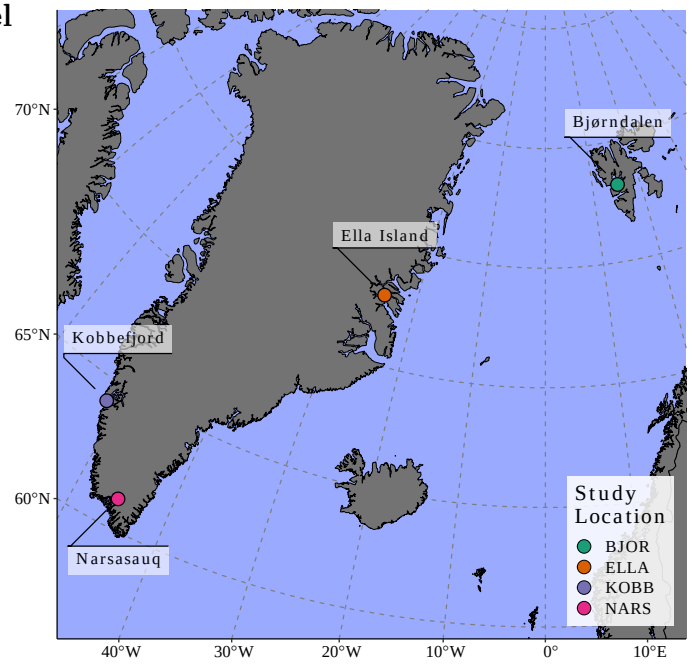


Figure 1: Map of the four study locations. Annotated labels are the english or local vernacular names for the locations, while the color legend names are the acronyms for the sites used throughout this report. Created in R using `sf`, `rnaturalearth` & `ggforce` (Pebesma 2018; Pedersen 2021; South 2017).

can be understood as a method that balances the amount of learning the model is able to derive from each class. The purpose of Inverse Class Frequency Weighted Loss is to ensure that the model does not become more attuned to high-frequency classes, which could lead to a model which is biased towards high-frequency classes. The full technical details of training `YOLOFlower` can be found in (Svenning and Larsen 2022) which is included in the appendix and code can be reviewed in the YOLOFlower repository.

2.2 Training dataset

The training dataset consists of $\sim$ 2400 images which were manually annotated by Rebekka Ween (Ween et al. 2022) across 21 time-series and 2 locations (Narsaasauq and Bjørndalen). The healthiest individuals were chosen for manual annotation, so as to maximize the information in each manually reviewed image (Ween et al. 2022), a choice which could cause bias issues in traditional machine learning workflows, but which we were able to work around and which may even have benefitted the performance of the `YOLOFlower` model (Svenning and Larsen 2022). I mostly attribute the lack of bias in our model to the previously described Inverse Class Frequency Weighted Loss, however this method will essentially decrease the learning rate of the model if the classes are unbalanced, as they naturally are in *S. acaulis* , thus by cherrypicking timelapse series of cushions with larger numbers of fruits, the model will be able to learn more effectively.

2.3 Object Tracking

The output of the `YOLOFlower` model on a single image is the inferred bounding boxes along with the class (lifestage) identities of each. However in combination with one of the post-hoc tracking algorithms found in the Multiple Object Tracking literature (Han et al. 2020; Wang et al. 2020; Zhang et al. 2021) it is possible to connect the predicted bounding boxes between images producing what is known as *tracks*. In this project each track corresponds to an individual *S. acaulis* flower. For the purpose of this project I have chosen `ByteTrack`, as it is both highly performant as well as simple to work with (Zhang et al. 2021). The algorithm behind `ByteTrack` was developed for purposes such as pedestrian tracking on traffic cameras, and is both theoretically well-founded

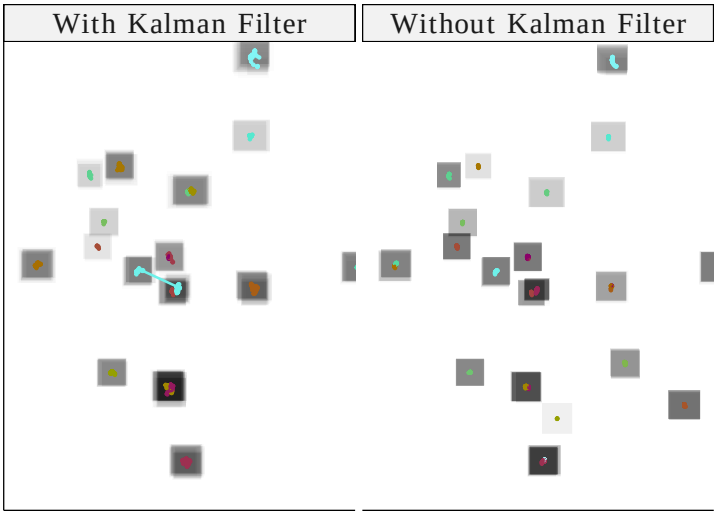


Figure 2: Example of bounding box drifting with and without the use of the Kalman filter. Colors represent different individuals, while the axes are bounding box centroid in image space. A randomly selected timelapse (from Bjørndalen 2020) was used as a representative example; however, the same pattern can be seen for the remaining timelapse series, with many showing even more signs of bounding box drifting.

as well as proven in the real world (Zhang et al. 2021). However, in the case of tracking flowers on stationary timelapse cameras, the algorithms has a major flaw, which is build into it on purpose; it assumes that the objects are moving and predicts their future positions using the widely used Kalman filter (Zhang et al. 2021). This causes serious undesirable drifting of the bounding boxes; however, I found that simply omitting the use of the Kalman filter (i.e. trajectory/speed predictions) reduced drifting considerably (Figure 2). A second issue with using `ByteTrack` in this project is that `YOLOFlower` has (rightly) learned different bounding box size distributions for the different classes, and since `ByteTrack` uses the IoU (Intersection over Union) metric for matching bounding boxes (Zhang et al. 2021), class transitions are associated with instance disassociation (that is failure to match bounding boxes between images correctly). In this project I aim to partly mitigate this by using the simpler centroid distance metric for bounding box matching. Both of the issues with `ByteTrack` described in this section are in fact already recognized by the authors of `ByteTrack`, however currently no single best solution exists and instead caution must be taken in choosing whether to use the Kalman Filter and which similarity metric is best for the data in question (Zhang et al. 2021). Although this method already has great performance at an acceptable inference speed (0.5 fps), further work, such as Mann (2022), on semi-stationary object tracking in timelapse series could most likely provide significant performance and efficiency improvements.

2.4 Image Alignment

Images are aligned using a simple sequential procedure:

1. Skip alignment of the first image (as there is no previous image to align with) and save it as the first "previous image".
2. For every other image subset both the previous and current image by a factor of 4.
3. Calculate the alignment transformation matrix between the images using the `imreg_dft` python module's `similarity` function (Reddy and Chatterji 1996).
4. Transform the current image using the previously calculated transformation matrix using the `transform_img` function from the `imreg_dft` module (Reddy and Chatterji 1996).
5. Return to step 2

This alignment procedure was chosen based on its simplicity and speed. Although selected on the basis of speed, the total inference time still increased by a factor 2-4× with image alignment enabled. Therefore, I suggest that future work on similar tasks include the image alignment step as a part of the tracking algorithm, something which is already well-known in the Multiple Object Tracking literature (Han et al. 2020).

2.5 Inference dataset

The full timelapse image dataset consists of images taken with 10 second intervals, this is a much temporal frequency than necessary for lifestage analysis of flowers of *S. acaulis* as the transitions between different stages happen over hours to days, not seconds. Under this consideration as well as to limit the computational resource use and significantly shorten the full inference time, I opted to each timelapse to a minimum time interval between images of 1 hour, instead of 10 seconds, reducing the number of images to "only" 37 173 across 65 different time-series.

2.6 Image Pipeline

The details of the different steps involved in the image pipeline developed for this project, that is the prodedure of turning images into lifestage timelines, are connected in a complex pipeline for which an illustrative overview is given in Figure 3. The initial step in the pipeline is collecting the images, this was done prior to the start of this project over the period 2019-2021 across the four study locations. As a result of this fieldwork Toke T. Høje and team amassed a large dataset of images, in the order of 10 million images. From these images a small subset was cherrypicked such that it included only highly productive cushions which were then subset to a relatively "low" temporal resolution (6 or 24 hours), these images were then manually annotated by Rebekka Ween (Ween et al. 2022) and formed the baseline for the `YOLOFlower` model (Svenning and Larsen 2022). In the `YOLOFlower` project the `SAHI` and `YOLOv5` projects were modified, combined and trained on the previously discussed manually annotated dataset (Akyon et al. 2022; Jocher et al. 2022; Svenning and Larsen 2022). Finally, in this project I subset the full image dataset to a maximum temporal resolution of 1 hour, and then deployed a sequential image alignment procedure described in Section 2.4 followed by inference using the trained `YOLOFlower` model. The result of inference on a single image is simply a set of bounding boxes with associated location, size, confidence and class label. The confidence scores are not used further in this project for simplicity, however they are an integral part of training as well as inference

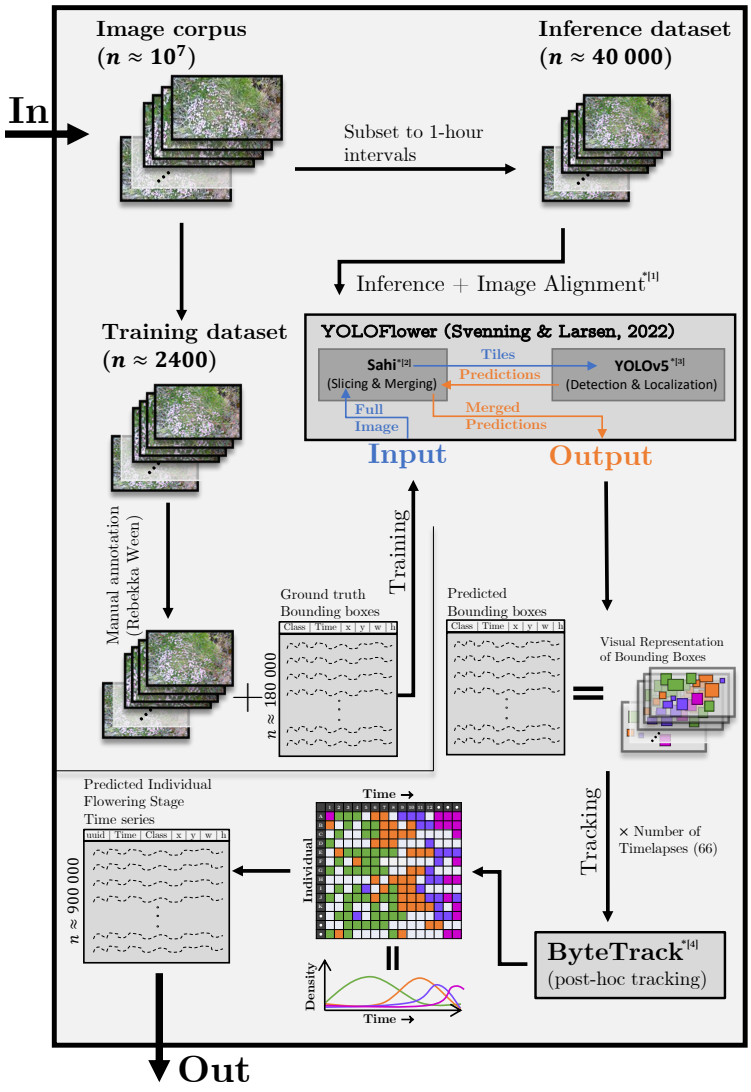


Figure 3: Lifestage Detection and Tracking Pipeline. This figure shows the overall workflow involved in obtaining flower stage tracking predictions from timeseries. Training and inference are done separately.
[1] See Section 2.4.
[2],[3] See Section 2.1.
[4] See Section 2.3.

using `YOLOFlower` (see Section 2.1). The bounding boxes were then matched between images using the modified online post-hoc algorithm `ByteTrack` (Zhang et al. 2021), allowing bounding boxes to be uniquely matches to an individual through time. All steps except training of `YOLOFlower` are computed online, i.e. on-the-fly. Using the described pipeline i produced a set of 64 862 individual flowers tracked over an average of $\sim$ 47 hours (median $\approx$ 8 hours, tracking times are extremely left-skewed with a long tail) for a total of 897 320 flower/fruit detections as well as 18 919 null-detections (images without any flowers/fruits).

2.7 Weather Data

Weather data was scraped from the `api.weather.com` API using a custom `R` (R Core Team 2022) script. The weather stations canonical API names used for each of the locations are: Narsarsuaq Airport (Narsarsuaq/NARS), Svalbard (Bjørndalen/BJOR), Godthåb (i.e. Nuuk) Airport (Kobbesund/KOBB) and Nerlerit Inaat (Ella Island/ELLA).

All weather stations except Nerlerit Inaat are close enough to the study site as to not be a concern; however, the distance between Nerlerit Inaat and Ella Island is about 250 km. The altitude of the imaged *S. acaulis* cushions is a significant, but not included, confounding factor on the temperature. In future work altitude of the cushions or local temperature measurements should be used to eliminate any possible bias introduced by omission.

2.8 Ecological Analyses

All ecological analyses and figures were made using the `R` programming language and especially the `tidyverse` framework (R Core Team 2022; Wickham et al. 2019). All parametric and semi-parametric models are created using the `mgcv` package (Wood 2011).

2.8.1 Temperature and Within-Cushion Flowering Period Variability

In order to assess the relationship between within-cushion flowering period variability I have chosen to define the flowering period of a set of data-points to simply be the between the 20-30% and the 70-80% day of the year quantile of the observed flowering time stamps: For a given set of observed flowering time stamps t the flowering period, L is:

$$L = \mathrm{Q}(t, 1 - qm) - \mathrm{Q}(t, qm) \tag{1}$$

where qm is a value between 0.2-0.3 and $Q(x, y)$ gives the y^{th} quantile of x. The quantile range is used to limit the effect of spurious classifications. Within-cushion flowering variability, LV, was quantified as the fraction between the flowering period of the entire cushion, $L_{cushion}$, over the flowering period of the individual flowers within the cushion:

$$LV = \frac{L_{cushion}}{\overline{L_{individual}}} \tag{2}$$

where $L_{cushion}$ is simply calculated using Equation 1, while $\overline{L_{individual}}$ is the mean of the flowering period for each individual flower within the cushion weighted by the number of frames the given individual has been tracked over (under the assumption that longer tracks are associated with less fragment tracks and thus provide better estimates of the individual flower flowering period).

To assess the relationship between temperature and within-cushion flowering variability I have chosen to model within-cushion flowering variability as a log-linear relationship with mean daily temperature over the entire cushion flowering period with study site as a random effect. Parameter estimates are fitted using a Quasipoisson generalized linear mixed model (GLMM), while parameter confidence intervals are calculated by nested bootstrapping (level 1: cushion, level 2: individual) as well as random uniform sampled quantile-cutoff (qm). The choice of distribution for the GLM is based on two considerations; (1) length of the flowering period is a count variable and could thus reasonably be expected to be either Poisson or negative binomial distributed, however since LV is a ratio between two integer distributions whose independence is highly unlikely, the distribution of LV is essentially unknown. (2) Manual inspection of heteroscedacity and residual distribution of a test-model on the full dataset using a number of different distributions; gaussian (OLMM), quasipoisson and negative binomial. The considered distributions were limited to those compatible with the `bam` function in the `mgcv` which allows both flexible and fast estimation of GLMM when the number of random effects is moderate (Wood 2011; Wood et al. 2017).

The goal of this statistical procedure was to relax the assumptions and improve the robustness of the analysis. In general the statistical procedure can be described in the following steps:

1. Bootstrap the cushions within each study site.
2. Bootstap the individuals within each cushion.
3. Calculate the flowering period of all individuals and cushions using Equation 1.
4. Summarize the within-cushion variability of all cushions using Equation 2.
5. Estimate the parameter values of the quasipoisson distributed GLMM with the formula $LV \sim \overline{T} + (1 \mid \text{Study Site})$.
6. Repeat from step 1 n times.
7. Calculate parameter confidence intervals as the 2.5%-97.5% quantiles of the n estimates for each parameter (intercept and temperature effect) and p-values after North et al. (2002).

2.8.2 Fruiting Setting Success

To analyze the relationship between flowering period length and mean daily flowering period temperature I have chosen to assign an individual as successfully fruit setting if it has been classified as a mature fruit in more than 10 images (timelapse frames). Daily temperature is calculated as the midpoint between minimum and maximum daily temperature, and the mean is taken over all days from the first flowering day to the last.

Flowering success is then modelled using a binomial generalized additive mixed model (GAMM) with flower period length or flowering period mean temperature as seperate additive predictors with study site, year and cushion as nested random effects, which should limit the confounding effect of cushion altitude discussed in Section 2.7. The total number of frames as well as the number of days an individual is tracked, i.e. track length, are also included as additive effects, since these represent significant confounding factors.

3 Results

3.1 Inferred Annual Lifestage Distribution

The result of the flower detection pipeline consists of 897 320 *S. acaulis* flower lifestage detections, consisting of each detected individual flower/fruit in each picture excluding the 18 919 null-predictions, one for each image with no detected flowers/fruits. The frequencies of each lifestage for each cushion in two-day intervals is visualized in Figure 5 to provides an overview of the data, while the overall temporal lifestage density distribution, i.e. flowering and fruiting phenology, is estimated and shown in Figure 4.

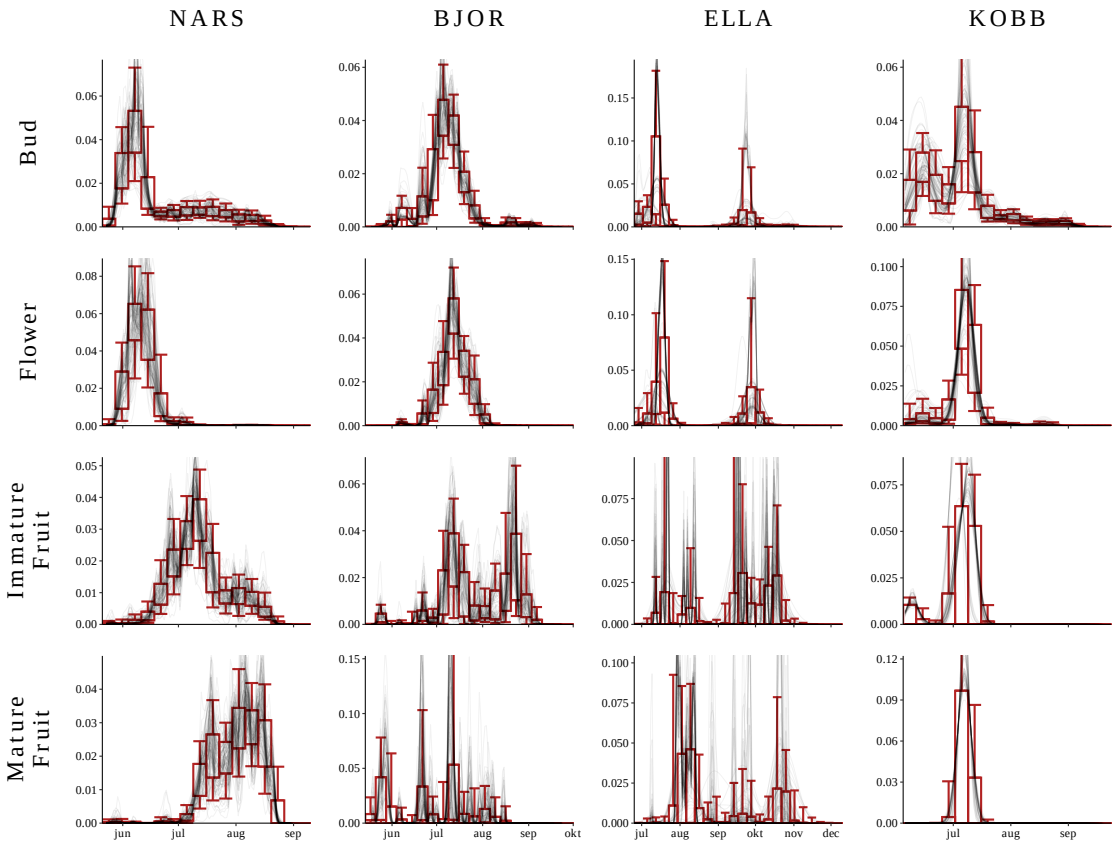


Figure 4: Bootstrap density estimates of annual lifestage frequencies across the different study locations (NARS: Narsasauq, BJOR: Bjørndalen, ELLA: Ella Island, KOBB: Kobbesund). To estimate the uncertainty in the kernel density estimates a two-level (level 1: cushion, Level 2: individual) nested bootstrap analysis ($n = 100$) is performed. Individuals are resampled with weight equal to the square root of the length of their time series under the assumption that shorter tracks are associated with higher degrees of detection errors. The semi-transparent lines show the individual bootstrap density estimates, while the red boxplots show aggregated 0.05%-[0.25%, 0.75%]-0.95% quantiles in one-week intervals.

The variability in the temporal density shown in Figure 4 is likely the result of a complex interaction of the differing climates/habitats and sampling across the different locations combined with model uncertainty in the flower detection and lifestage classification and perhaps generalization error, due to the `YOLOFlower` model only being trained on images from Bjørndalen and Narsasauq (Svenning and Larsen 2022). Although I have not been able to quantify the magnitude of the generalization error, I have conducted a manual review of a small number of random images across the different study sites and years, with no discernible difference in classification or localization error. The large variability in Figure 4 especially on Ella Island and in the immature and mature fruit class can thus most likely be attributed to small sample sizes and poor fruiting seasons. However, an important secondary explanation for the variability that is unique to Ella Island is that the timelapse monitoring period differed significantly between the two study years; (2020) september-november, (2021) july-august, which explains the bimodal annual flowering distribution displayed for Ella Island in Figure 4. Finally, the large holes in the timelapse images as can be seen in Figure 5, which occurs indiscriminately across the monitoring period also bias the phenology estimates and future work should either model the holes explicitly or use an imputation strategy. A particularly problematic example is found in the 2020 Narsasauq (NARS) timelapse series, where a 1-2 week period of images are missing for all cushions in the middle of the flowering period.

3.2 Temperature and Within-Cushion Flowering Variability

I did not find a significant relationship between temperature and within-cushion flowering variability ($p \approx 0.59$), based on a robust nested bootstrap approach. However, the quantile confidence interval (Table 1) barely contains zero and is heavily skewed towards negative estimates of the relationship, the expectation of the proposed hypothesis. Although I cannot able to reject the null hypothesis, it suggests that the cushion sample size in this project is slightly too small, and even a slightly increased sample size might resolve the relationship. Although this study provides a large amount of microlevel detail, the number of study sites is only four with each study site being visited for one, two, two and three years totalling "only" 65 different cushions. When combining autocorrelation within cushions, study sites and years, the effective sample size of this study must be considered to be lower than the total number of cushions (65).

Estimate	**2.5%-CI**	**97.5%-CI**	**Empiric P-value**
−0.09156	−0.44474	0.00642	0.059

Table 1: Bootstrap results for the relationship between temperature and within-cushion variability. The estimate is the median slope of the bootstrap slope distribution between mean flowering period temperature and the increase in flowering period length of the pooled individuals within a cushion over the mean flowering period length of the unpooled individuals within a cushion. The relationship is modelled using a quasipoisson GLMM with study site as a random effect.

3.3 Fruiting Success

I found a highly significant, but nonlinear, relationship between mean daily flowering period temperature and fruiting success probability (see Table 2), with lower probabilities at low temperatures

followed by a increasing probabilities up until intermediate temperatures where the probability decreases between the temperatures 5-10 C° before increasing again. It should be noted however that more than 95% of individuals have a mean daily flowering period temperature greater than 5 C°, which means that for almost all individual flowers increasing temperatures lead to increasing fruiting probabilities. Length of the flowering period was also found to have a marginally significant effect on fruiting probability following a inverted parabola shape with higher probabilities at intermediate flowering period lengths ($\sim$ 40 days). The two included confounding factors, track length and number of frames observed in, are both highly significant and show a positive monotonic relationship with fruiting success probability, suggesting that track fragmentation might be negatively correlated with fruiting success, since short track lengths as measured in date ranges or number of frames can reasonably be expected to correspond to fragmented tracks. This relationship could also be explained by ecological processes, however in this project it is not possible to disentangle the confounding effect track disassociation.

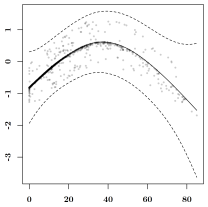
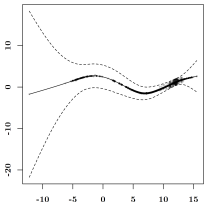
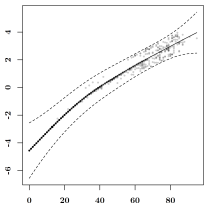
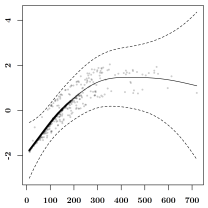
Parameter	Estimated DF	χ^2	P-value	Shape
Length of Flowering Period (Days)	2.58	8.44	0.045	
Mean Daily Flowering Period Temperature (C°)	3.83	14.85	$7.7 \cdot 10^{-3}$	
Tracking Period Length (Days)	2.91	61.38	$< 10^{-15}$	
Number of Frames Observed	2.68	28.40	$7.1 \cdot 10^{-6}$	

Table 2: Estimated degrees of freedom, significance and relationship with successful fruiting probability. See Section 2.8.2 for more details. Shape plots are produced using `plot.gam` with partial residuals and confidence intervals after Marra and Wood (2012). The x-axis the values of predictor in question and the y-axis is the partial linear predictor.

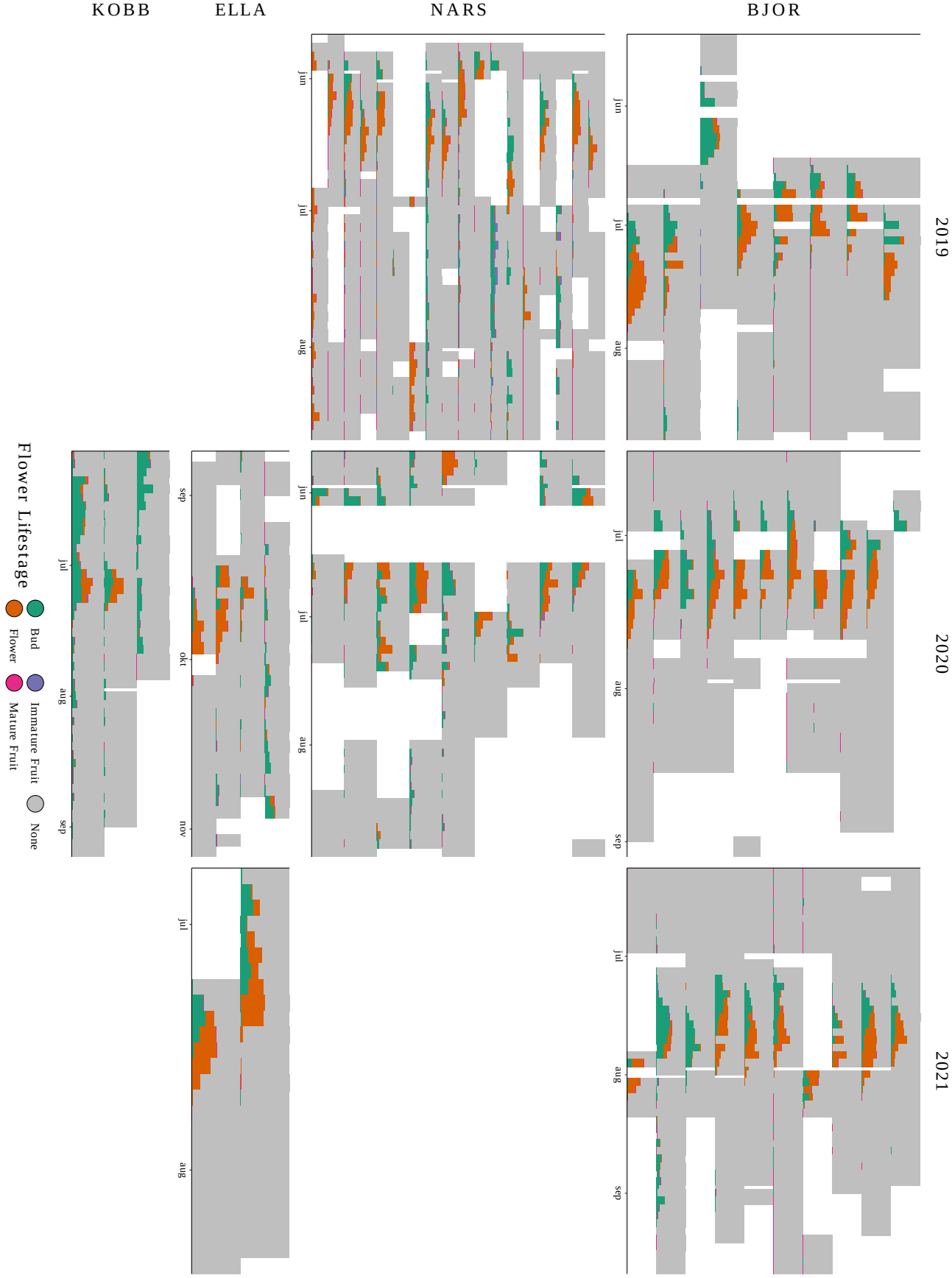


Figure 5: Lifestage histograms in 2-day intervals. Each row in the subfigures correspond to a specific *S. acaulis* cushion, and the area of the colored regions correspond to the proportion of different classes in the given 2-day interval out of the maximum observed number of flower-individuals within the cushion across all time intervals. The colored regions areas correspond to time intervals with detected flowers/fruits, while the gray areas represent intervals with images but without detected flowers/fruits (null-predictions) and white areas are time intervals without images.

4 Discussion

In this project I have shown how deep learning frameworks such as `Sahi` and `ByteTrack` (Akyon et al. 2022; Zhang et al. 2021) can be adapted to fit the needs of biological research in which automatic monitoring plays an increasingly important role. However, at the same time I have found that prerequisite knowledge (linear algebra and matrix calculus) required for tinkering with the inner workings of deep learning models such as `YOLOv5`, can represent a significant barrier. The difference in the ease of adaptability of these frameworks can mostly be attributed to two factors (1) the nature of the code, is it low-level deep learning code, e.g. `PyTorch` code in the `YOLOv5` repository (Jocher et al. 2022; Paszke et al. 2019), or high-level abstract code, e.g. `SAHI` or `ByteTrack` (Akyon et al. 2022; Zhang et al. 2021), and (2) documentation and use of clean code principles (intuitive variable/function names and comments).

Guided by these two factors I recommend that biologists tasked with building an automated monitoring pipeline should focus on adapting the higher-level frameworks and attempt to avoid spending too many resources on tinkering with low-level deep learning model code. An analogy to traditional statistics/machine learning can be provided; most biologists will not change the model fitting code for a statistical model (e.g. Generalized Linear/Additive Mixed Models, Random Forest or Support Vector Machines), however it is relatively common to tinker with different response transformations or using bootstrapping or boosting for variability estimation, as I have done in this project. This should not however be taken as a recommendation against any tinkering with the inner workings of deep learning models, but instead as a caution that this will increase the required prerequisite knowledge considerable as well as the complexity of validating and ensuring reproducibility of the pipeline.

Using these principles I was able to produce large-scale high-frequency spatiotemporal data on the flowering and fruiting phenology of *S. acaulis* which under careful statistical considerations was used for ecological analyses of the flowering cycle. Although I was not able to confirm the proposed hypothesis on the effect of temperature on within-cushion flowering timing variability, the result was highly skewed in favor of larger within-cushion flowering phenology variation in colder habitats. Combined with the relatively small number of cushions examined, I expect that any possible future research which includes a slightly larger number of cushions will most likely confirm the proposed hypothesis. Furthermore the inclusion of cushion altitude, which was not included in this project as well as further environmental factors, all of which are relatively readily available, would also lead to more sound results.

I also show that the apparent fruit setting optimum flowering length of *S. acaulis* is approximately 40 days, while higher flowering temperature generally are associated with higher probabilities of fruit setting (Table 2). Following Wheeler et al. (2015) it is unsurprising that higher flowering temperature lead to greater fruiting success, given that flowering temperature can reasonably be expected to be negatively correlated with frost during flowering, which in turn could lead to frost-damage and unsuccessful fruiting. Plant-pollinator relationships could also provide a mechanism for the relationship with both temperature, as pollinators might be affected by temperature, and flowering length, where longer flowering would naturally lead to a higher probability of one or more pollination event(s), while very long flowering periods might be associated with flowers which were never successfully pollinated. Disaggregating pollinator and physiological effects would reasonably be possible by also detecting pollinators as was done in Mann (2022).

However, even considering the challenges outlined above, I was able to show how successful fruit setting in *S. acaulis* as expected is affected by flowering length and temperature, providing an example of how the explosive growth in the field of deep learning can be used in conjunction with classical ecological statistics to expand range and scale of research possibilities in time and resource constrained endeavours.

5 Acknowledgments

This work would not have been possible without the willingness of my supervisors, Toke T. H., Hjalte M. R. M. and Phillip F. T., to allow me to undertake such a large and risky interdisciplinary project. Without their guidance this project would not have been possible. Special thanks to Rebekka W. for her invaluable work in manually annotating the images, which provided the very foundation for the `YOLOFlower` model as well as Toke T. H. for providing me with access to the timelapse imagery without which there would be no project at all.

6 Supplementary Materials

All scripts and data for the ecological analyses and figures in this report can be found at https://github.com/asgersvenning/automaticSileneAcaulisMonitoring. The training script for `YOLOFlower` and the *S. acaulis* tracking pipeline described in this report can be found at https://github.com/satansju/YOLOFlower.

7 References

Akyon, F. C., Onur Altinuc, S., & Temizel, A. (2022). Slicing Aided Hyper Inference and Fine-Tuning for Small Object Detection. *2022 IEEE International Conference on Image Processing (ICIP)*, 966–970. https://doi.org/10.1109/ICIP46576.2022.9897990

Baqui, P. O., Marra, V., Casarini, L., Angulo, R., Diáz-Garciá, L. A., Hernández-Monteagudo, C., Lopes, P. A., López-Sanjuan, C., Muniesa, D., Placco, V. M., Quartin, M., Queiroz, C., Sobral, D., Solano, E., Tempel, E., Varela, J., Vílchez, J. M., Abramo, R., Alcaniz, J., … Taylor, K. (2021). The miniJPAS survey: Star-galaxy classification using machine learning. *Astronomy and Astrophysics*, *645*, A87. https://doi.org/10.1051/0004-6361/202038986

Burhan, S. A., Minhas, D. S., Tariq, D. A., & Nabeel Hassan, M. (2020). Comparative Study of Deep Learning Algorithms for Disease and Pest Detection in Rice Crops. *Proceedings of the 12th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2020*. https://doi.org/10.1109/ECAI50035.2020.9223239

CaraDonna, P. J., Iler, A. M., & Inouye, D. W. (2014). Shifts in flowering phenology reshape a subalpine plant community. *Proceedings of the National Academy of Sciences of the United States of America*, *111*(13), 4916–4921. https://doi.org/10.1073/PNAS.1323073111/SUPPL_FILE/ST07.DOCX

Christin, S., Hervet, É., & Lecomte, N. (2019). Applications for deep learning in ecology. *Methods in Ecology and Evolution*, *10*(10), 1632–1644. https://doi.org/10.1111/2041-210X.13256

Dietrich, L., & Körner, C. (2014). Thermal imaging reveals massive heat accumulation in flowers across a broad spectrum of alpine taxa. *Alpine Botany*, *124*(1), 27–35. https://doi.org/10.1007/S00035-014-0123-1/FIGURES/6

Ferreira, A. C., Silva, L. R., Renna, F., Brandl, H. B., Renoult, J. P., Farine, D. R., Covas, R., & Doutrelant, C. (2020). Deep learning-based methods for individual recognition in small birds. *Methods in Ecology and Evolution*, *11*(9), 1072–1085. https://doi.org/10.1111/2041-210X.13436

Gehring, J. L., & Delph, L. F. (1999). Fine-scale genetic structure and clinal variation in Silene acaulis despite high gene flow. *Heredity 1999 82:6*, *82*(6), 628–637. https://doi.org/10.1046/j.1365-2540.1999.00524.x

Gupta, A., Anpalagan, A., Guan, L., & Khwaja, A. S. (2021). Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, *10*, 100057. https://doi.org/10.1016/J.ARRAY.2021.100057

Han, S., Huang, P., Wang, H., Yu, E., Liu, D., & Pan, X. (2020). MAT: Motion-Aware Multi-Object Tracking. *Neurocomputing*, *476*, 75–86. https://doi.org/10.48550/arxiv.2009.04794

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., TaoXie, Michael, K., Fang, J., Imyhxy, Lorna, Wong, C., Yifu, Z., V, A., Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, … Xylieong. (2022). *ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations*. Zenodo. https://doi.org/10.5281/zenodo.7002879

Kennamer, N., Kirkby, D., Ihler, A., & Sanchez-Lopez, F. J. (2018). ContextNet: Deep learning for Star Galaxy Classification. https://proceedings.mlr.press/v80/kennamer18a.html

Liu, L., Wang, R., Xie, C., Yang, P., Wang, F., Sudirman, S., & Liu, W. (2019). PestNet : an end-to-end deep learning approach for large-scale multi-class pest detection and classification. *IEEE Access*, *7*, 45301–45312. https://doi.org/10.1109/ACCESS.2019.2909522

Mann, H. M. R. (2022). Automatic monitoring of plant-pollinator interactions with computer vision and deep learning. *PhD Thesis. Aarhus University, Department of Ecoscience, Denmark.*

Marra, G., & Wood, S. N. (2012). Coverage Properties of Confidence Intervals for Generalized Additive Model Components. *Scandinavian Journal of Statistics*, *39*(1), 53–74. https://doi.org/10.1111/J.1467-9469.2011.00760.X

Norouzzadeh, M. S., Morris, D., Beery, S., Joshi, N., Jojic, N., & Clune, J. (2021). A deep active learning system for species identification and counting in camera trap images. *Methods in Ecology and Evolution*, *12*(1), 150–161. https://doi.org/10.1111/2041-210X.13504

Norouzzadeh, M. S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M. S., Packer, C., & Clune, J. (2018). Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences of the United States of America*, *115*(25), E5716–E5725. https://doi.org/10.1073/PNAS.1719367115/SUPPL_FILE/PNAS.1719367115.SAPP.PDF

North, B. V., Curtis, D., & Sham, P. C. (2002). A note on the calculation of empirical P values from Monte Carlo procedures. https://doi.org/10.1086/341527

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury Google, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Xamla, A. K., Yang, E., Devito, Z., Raison Nabla, M., Tejani, A., Chilamkurthy, S., Ai, Q., Steiner, B., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.

Pebesma, E. (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, *10*(1), 439–446. https://doi.org/10.32614/RJ-2018-009

Pedersen, T. L. (2021). *ggforce: Accelerating 'ggplot2'*. https://cran.r-project.org/package=ggforce R package version 0.3.3

Peterson, M. L., Doak, D. F., & Morris, W. F. (2018). Both life-history plasticity and local adaptation will shape range-wide responses to climate warming in the tundra plant Silene acaulis. *Global Change Biology*, *24*(4), 1614–1625. https://doi.org/10.1111/GCB.13990

Post, E. S., Pedersen, C., Wilmers, C. C., & Forchhammer, M. C. (2008). PHENOLOGICAL SEQUENCES REVEAL AGGREGATE LIFE HISTORY RESPONSE TO CLIMATIC WARMING. *Ecology*, *89*(2), 363–370. https://doi.org/10.1890/06-2138.1

R Core Team. (2022). R: A Language and Environment for Statistical Computing. https://www.r-project.org/

Reddy, B. S., & Chatterji, B. N. (1996). An FFT-based technique for translation, rotation, and scale-invariant image registration. *IEEE Transactions on Image Processing*, *5*(8), 1266–1271. https://doi.org/10.1109/83.506761

Simhambhatla, R., Okiah, K., & Slater, R. (2019). Self-Driving Cars : Evaluation of Deep Learning Techniques for Object Detection in Different Driving Conditions. *SMU Data Science Review*, *2*(1), 1–27. https://scholar.smu.edu/datasciencereview/vol2/iss1/23

South, A. (2017). *rnaturalearth: World Map Data from Natural Earth*. https://cran.r-project.org/package=rnaturalearth
R package version 0.1.0

Sun, Y., Liu, X., Yuan, M., Ren, L., Wang, J., & Chen, Z. (2018). Automatic in-trap pest detection using deep learning for pheromone-based Dendroctonus valens monitoring. *Biosystems Engineering*, *176*, 140–150. https://doi.org/10.1016/J.BIOSYSTEMSENG.2018.10.012

Svenning, A., & Larsen, S. S.-Y. (2022). YOLOFlower Near-Real Time flowering stage detection.

Waddle, E. (2017). *Factors Driving Flowering Phenology and Reproductive Success in Silene acaulis and implications for response to climate change* (Doctoral dissertation). University of Colorado at Boulder.

Wang, Z., Zheng, L., Liu, Y., Li, Y., & Wang, S. (2020). Towards Real-Time Multi-Object Tracking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12356 LNCS*, 107–122. https://doi.org/10.1007/978-3-030-58621-8_7

Ween, R. E., Yoccoz, N., Høje, T. T., & Eidesen, P. B. (2022). Timing is everything: Within-plant flowering phenology impacts fruit production in the Arctic-Alpine cushion plant *Silene acaulis* (L.) Jacq. *Masters of Science in Biology - Northern Populations and Ecosystems*.

Wheeler, H. C., Høye, T. T., Schmidt, N. M., Svenning, J. C., & Forchhammer, M. C. (2015). Phenological mismatch with abiotic conditions—implications for flowering in Arctic plants. *Ecology*, *96*(3), 775–787. https://doi.org/10.1890/14-0338.1

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Wood, S. N. (2011). Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society (B)*, *73*(1), 3–36.

Wood, S. N., Li, Z., Shaddick, G., & Augustin, N. H. (2017). Generalized Additive Models for Gigadata: Modeling the U.K. Black Smoke Network Daily Data. *Journal of the American Statistical Association*, *112*(519), 1199–1210. https://doi.org/10.1080/01621459.2016.1195744/SUPPL_FILE/UASA_A_1195744_SM4279.ZIP

Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., & Wang, X. (2021). ByteTrack: Multi-Object Tracking by Associating Every Detection Box, 1–21. https://doi.org/10.48550/arxiv.2110.06864

Appendix

A YOLOFlower

The report on `YOLOFlower` by myself (Asger Svenning) and Simon Sataa-Yu Larsen (Svenning and Larsen 2022), is included for reference below as it is unpublished.

YOLOFLOWER
NEAR-REAL TIME FLOWERING STAGE DETECTION

Asger Svenning au644314
Simon Sataa-Yu Larsen au586654

December 9, 2022

Figure 1: Crop of a data image sample - Moss Campion

Contents

ABSTRACT

Object classification has become ubiquitous in the application of modern digital media, however the process of detecting and classifying objects is still done manually in almost all botanical and ecological field research. This problem is a subfield of Visual Recognition in the Deep Learning literature known as (Multiple) Object Detection. The subfield is rapidly evolving and currently a multitude of different approaches exist including the YOLO (You Only Look Once) framework among others.

In this course project, we solve flower detection and phenology state classification for flowers on individuals of the species Moss Campion (*Silene acaulis*) in a small dataset of stationary time lapse images using the YOLOv5 architecture. Producing an accurate and powerful Multiple Object Detection model on a small dataset with very limited variation such as the one presented here requires careful consideration.

We show that including an image tiling preprocessing step increases the model performance drastically, while inverse class frequency weighted loss results in a significantly less biased model.

1 Introduction

1.1 Motivation

Currently a major part of biological data collection consists of various human visual recognition tasks such as classifying species identities, species coverage and individual attributes (such as size, weight, age, life-stage, color etc.). Due to the tremendous workload associated with this type of work, the use of deep learning for automatic visual recognition is gaining increasing traction in the biological literature. However many of the tasks that are tackled by biologists, especially in the context of very specialised and niche field work, are often not amenable to out-of-the-box deep learning solutions, which we suggest can be explained mainly by three factors:

1. Biological visual tasks can be less suitable for for transfer learning due the large domain difference between human objects versus natural "objects".
2. Small data sets due to the prohibitive workload in obtaining a large enough data set for a often very specific tasks (e.g. detection and localization of flowers from a specific species, which may be rare, occur in hard-to-reach and/or science-resource scarce regions).
3. The data distribution is often unbalanced and has poor coverage of the underlying distribution, due to the nature of data collection in natural environments.

We have chosen to focus on the specific challenge of multiple object detection of flowers on individuals of the species Moss Campion, *Silene acaulis*, which is a plant species where individuals can display up into the hundreds of flowers (see Figure 1). This builds on previous work in a masters project by Rebekka Eriksen Ween (Ween et al., 2022), where the flowers were manually localized and annotated with the flowering stage. This task was essentially a manual multiple object detection task and the goal was chiefly to obtain the predictions, such that these could be used for biological-statistical analysis on the life cycles of the species. In this project we attempt to automate the annotation process which is otherwise extremely time consuming; In the case of the data set we are using, Rebekka E. W. has painstakingly manually drawn more than 800 000 bounding boxes and annotated each with the correct class label.

1.2 Dataset

The data set is provided by Toke T. Høje, Professor at Ecoscience at Aarhus University, and contains a set of top-down timelapse images ($N = 2,386$) of vegetation with annotated bounding boxes ($N = 855,084$) and labels. These images were taken using the camera **TimelapseCam Pro** in stationary timelapse image series ($N = 21$) with image intervals of either 6 or 24 hours across the flowering period of Moss Campion (i.e. from before flower-production begins, until after all fruits are gone). All images are taken top-down from a height of 60 cm centered above $\sim 1-2$ individuals of Moss Campion (e.g. Table 1). Thus any model trained on this data set will be highly likely to be limited to images taken in this specific manner.

We use pretrained weights from YOLOv5 (Jocher et al., 2022) as a basis for transfer learning. The weights have been pretrained on the MSCOCO dataset (Lin et al., 2014).

Figure 2: Cropouts of the target classes of the different phenology stages of Moss Campion. From: (Ween et al., 2022).

2 Related Work

In our project, we have chosen to use the YOLOv5 model for object detection. However, there are other models available such as the Single-Shot Detector (SSD) that have been shown to outperform YOLO for detecting smaller objects because it adds single-shot detection layers and the end of the encoder for predicting bounding boxes (Liu et al., 2015). Recent versions of the YOLO framework, such as YOLOx and YOLOv7, have also been published (Ge et al., 2021; Wang et al., 2022). There is an adaptation of YOLOv4 for real-time flower detection in mobile applications (Cheng and Zhang, 2020), which uses a training set with larger photos of vegetation (Hiary et al., 2018). However, this

solution only detects the species name of the flowers, not the state of the flower. While this work is relevant, it does not directly address the problem we are trying to solve, which is why we have proposed a solution based on YOLOv5.

3 Model Architecture and Implementation

3.1 You Only Look Once (YOLOv5)

The YOLO framework first presented in the 8 Jun 2015 and has laid ground work for a series of adaptations of the approach for real time object detection (Redmon et al., 2015). The newest version including a paper is YOLOv7 (Wang et al., 2022). For this project we have dived into the YOLOv5 framework, that has yet to release a paper (Redmon et al., 2015). We chose this framework over newer versions of the YOLO approach because it was available and easily adaptable for our purpose. Namely detection of flowering stages.

The YOLO approach consists of a single convolution neural network that has the capability of both processing images in real size, predict bounding boxes and class probabilities of the the processed image in a single evaluation. This enables the entire network to be optimised on detection performance. The architecture of the network consists of an encoder and a decoder. An illustration of the encoder can be found in the appendix D.1.

Instead of doing sliding windows, when predicting bounding boxes, the YOLO framework uses anchors to generate potential bounding boxes which precision is evaluated in every run, making the YOLO framework fast and efficient at detecting objects.

The network processes complete images and it combines the class prediction, generation and evaluation of bounding boxes across all classes simultaneously. Thus the network can reason about the full image and all the objects in it globally. So when training we don't need to train the networks capability of predicting classes and the position of the bounding boxes separately. Thus we can train the complete network in one go.

When predicting bounding boxes, the system divides the input image into an $S \times S$ grid, and when the centre of an object falls into a grid cell, that grid cell is then responsible for predicting the object. All of the grids cells predicts B bounding boxes and confidence scores of each box. The system uses this confidence score to say how likely it is that a box contains an object and how accurate it thinks the box is. Confidence is defined as $P(Object) \cdot IoU^{truth}_{prediction}$. $P(object)$ is the probability of the object being of that class and $IoU^{truth}_{prediction}$ is the intersection over union of the predicted bounding box and the ground truth. If there is no object in a bounding box, then the confidence should be zero. In section 3.2.2 we will elaborate on these terms, and use the notation $l_{n,c} \cdot \mathcal{L}_{CIoU}$

A threshold is used for deciding, if the confidence level of a bounding box is high enough to keep creating predictions for each bounding box. If a bounding box has confidence level less than the threshold it is dropped. There is also a mechanism that filters out bounding boxes that are similar in both size and class predicted, such that the same object is not detected several times.

Furthermore YOLO uses a class probability map to give each region of a picture a score, that maps pixels of the picture to a class probability. Meaning how likely it is that those pixels represent each class or the background. The network predicts a class probability and an offset for each bounding box. Bounding boxes above a certain threshold is used to predict a class and the location of an object.

One downside with the YOLO architecture when the purpose is object detecting of smaller objects, is that the network uses fixed anchors, that is different positions for bounding boxes. This causes the network to have some spatial constraints for how it can place smaller bounding boxes resulting in lesser performance.

3.2 Class Imbalance

When doing multiple class object detection, a common problem is class imbalance. The class Imbalance problem occurs when there are several classes, that appear more frequent than other classes. The classes appearing most often is referred to as majority classes and the less frequent ones are referred to as minority classes. This skews the model to predict the majority classes better than the minority classes (Shrivastava, 2020).

If we were to over-sample the minority class or under-sample the majority classes, we would still not solve the problem of class imbalance. We might introduce new issues. Oversampling could lead to over-fitting and under-sampling could make the model miss key learning, as a result of removing samples from the data set.

A solution is sample weighting in the loss function. Our model uses a Binary Cross Entropy Loss function, that utilises the Sigmoid activation function.

3.2.1 Sigmoid Activation Function

To enable neural networks to predict non-linear shapes, activation functions are used. They come in several varieties, with either leaking or non leaking activation functions. In the YOLO framework, we use the Sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

3.2.2 Loss Functions

There are two loss functions used for the YOLOv5 network. One for the class predictions and one for predicting bounding boxes.

The class prediction loss function used in YOLOv5 is Binary Cross Entropy with Logits loss. This combines the Sigmoid function σ and the Binary Cross Entropy loss for numerical stability:

$$l_c(x, y) = L_c = \{l_{1,c}, \dots, l_{N,c}\}^T \tag{2}$$

$$l_{n,c} = -w_{n,c}(p_c y_{n,c} \log \sigma(x_{n,c}) + (1 - y_{n,c}) \log(1 - \sigma(x_{n,c})) \tag{3}$$

where, *x* is the predicted class label, *y* is the correct label, *c* is the class, *n* is the number of the samples in the batch, $w_{n,c}$ is the manual rescaling weight given to the loss of the n^{th} element of the batch and p_c is the weight of the positive answer for the class *c*. $w_{n,c}$ are the sample weights. We have constructed a simple heuristic for measuring the "information" in a sample, s, which we will use in the loss weights:

$$\omega = -\log|s| \cdot \sum_i^n \mathrm{P}(c_i|s) \cdot \log \mathrm{P}(c_i|s), \quad \mathrm{P}(c_i|s) = \frac{\#\text{ of Bounding boxes} \in c_i}{\text{Total }\#\text{ of Bounding boxes}} \text{ in sample } s \tag{4}$$

Where c_i is the i^{th} class, while $\mathrm{P}(c_i|s)$ is the frequency of the class in a sample. The weighting function is thus the product of the logarithm of the number of bounding boxes and the class entropy in a sample. We then combine this information criteria with inverse class frequency to obtain the final loss weighting term:

$$w_{n,c}(\gamma, \mu) = \frac{\{\omega_1, ..., \omega_n\}^\gamma}{(\text{Frequency of Class } c)^\mu} \quad \mu \in \{0, 1\} \tag{5}$$

Where γ and μ are weight modulating parameters for the sample and class weights respectively. In the implementation of YOLOv5, there is a problem with mismatch in the shapes of the tensors, when calculating the loss function. To solve this, either add empty dimensions and/or duplicate the tensors along the appropriate dimensions.

The network also uses Complete IoU Loss $\mathcal{L}_{CIoU}$ for improving the bounding boxes predicted by the network (Zheng et al., 2019). In the following loss function, b, b^{gt} is the centre of a predicted bounding box and the ground truth, meaning the correct bounding box for a certain object to be detected by the network. $B = (x, y, w, h), B^{gt} = (x^{gt}, y^{gt}, w^{gt}, h^{gt})$ are the coordinates for a bounding box, where x, y are the coordinate in the plane and w, h are the width and height respectively. The diagonal distance ρ is defined as the euclidean between the centre points of the bounding boxes.

$$IoU = \frac{|B \cap B^{gt}}{|B \cup B^{gt}|} \tag{6}$$

$$\mathcal{L}_{\text{CIoU}} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v, \tag{7}$$

where α is a parameter positive trade-off parameter, and v is a measurement of the consistency of the aspect ration given as:

$$v = \frac{4}{\pi^2} \left(\arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h} \right)^2 \tag{8}$$

3.3 Baseline

We have chosen the small version of YOLOv5 (YOLOv5s) for our experiments, as a compromise between performance and model size.

Our initial baseline model is a YOLOv5s model trained for 8 epochs on 80% of our data set, with the default hyperparameters as described in Section 3.3.1. This model is not converged, a choice we made to make the baseline more comparable to our experimental results, which are also based on unconverged models, where it proved inhibitive to train all the models to convergence due to resource and time constraints. In the results section (Section 5) we will compare our final model with a new baseline, that is trained to convergence.

Table 1: Baseline model performance

Class	P	R	$\mathrm{mAP}_{50\%}$	$\mathrm{mAP}_{50\text{-}95\%}$
Average	0.545	0.0814	0.0891	0.0285
Bud	1	0	0.0137	0.0096
Flower	0.524	0.275	0.339	0.112
Withered	0.201	0.132	0.106	0.295
Immature	1	0	0	0
Mature	1	0	0	0

3.3.1 Hyperparameters

To reduce the complexity and magnitude of this problem, such that it fits within the limits of this reports, we have opted to use the same hyper-parameters for all our experiments, except when explicitly mentioned or when they are changed as the parameter of interest in one of our experiments. The default hyperparameters are reported in Appendix B : Table B.1 for reproducibility.

3.3.2 Downsampling & Tiling

Downsampling the images using a naïve (i.e. non-learned) method, will necessarily result in a loss of information. Since our images original resolution 6040×3420 is way larger than the input resolution for YOLOv5 640×640 or 1280×1280 (larger models, not used in our report due to limited compute resources). We use slicing with a combination of downsampling. Meaning that we divide each individual picture into a grid of smaller tiles. Tiling uses an overlap parameter, that determines how much overlap is allowed $o \in \{0, 10\%, 20\%\}$. The amount of tiles produced by this operation is:

$$\mathrm{N}_{tiles} = \mathrm{N}_{images} \cdot \prod_{i}^{2} \left\lceil \frac{\mathrm{R_i}}{D \cdot I_i \cdot (1-o)} \right\rceil, \tag{9}$$

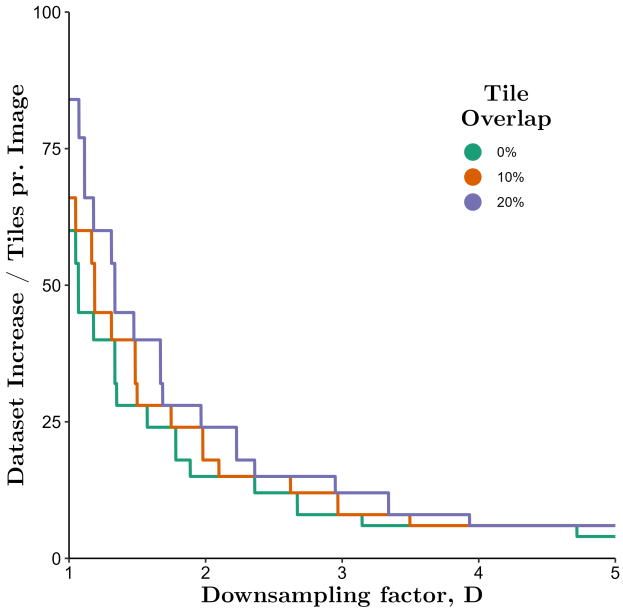
$$R = \{6040, 3420\}, \quad I = \{640, 640\} \tag{10}$$

And the increase in dataset is then:

$$\frac{\mathrm{N_{tiles}}}{\mathrm{N_{images}}} = \prod_{i}^{2} \left\lceil \frac{\mathrm{R_i}}{D \cdot I_i \cdot (1-o)} \right\rceil \tag{11}$$

Where o is the overlap size and R is the original image resolution and I is the tile resolution. This is also the number of tiles produced per image for a given D and o.

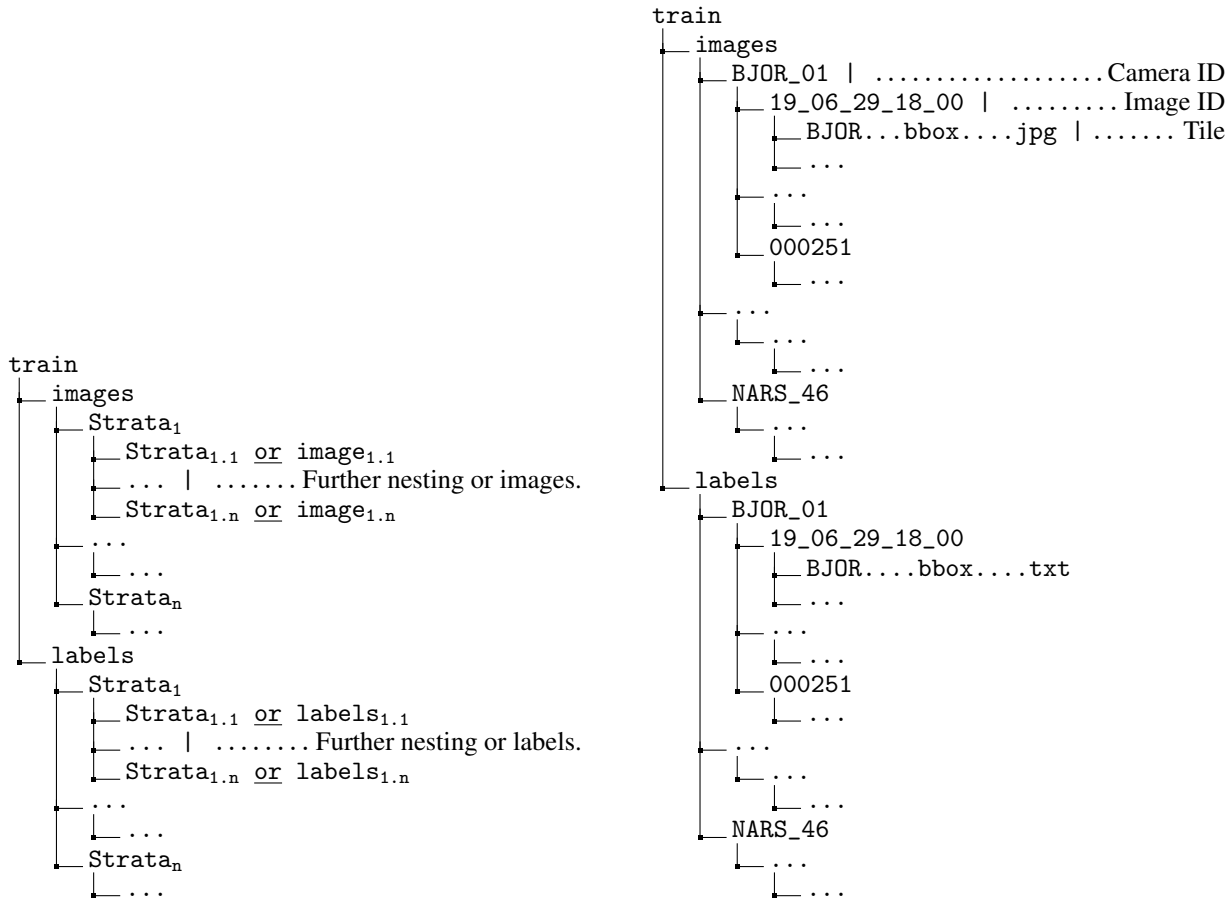
Figure 3: Dataset Increase as a Result of Tiling



We have opted to cull all tiles with zero labels for the training set, in order to somewhat counter the dramatic increases in dataset size, and with it increased training time lengths. This also increases the bounding box density, which might result in a more confident model. We leave the testing of the tile-culling as future work, due to project size restraints. The tiling code in our project is adapted from the SAHI library (Akyon et al., 2022), however the modified code is only for data set preprocessing before training such that the trained model weights can be used with the unmodified SAHI library.

3.4 Data leakage

Since our data set consists of 21 stationary timelapse image series, the visual contents of the images in a series is highly correlated. This extends to both the appearance, size and spatial patterns of the objects and bounding boxes. If the training and validation split was done randomly, as is often the default, this could lead to significant data leakage between the splits, invalidating the validation results. To address this issue we implemented a new folder structure which can be used to create more flexible train-validation splitting. To use our stratified data splitting feature the data set structure should be as follows:



(a) General folder structure

(b) Our tiled folder structure

Figure 4: Our reimplementation of the YOLOv5 folder structure.

Notice that we do not use the validation and test folders. Instead creating splits is now a class-function of our custom `LoadFlower` class which inherits from the YOLOv5 class `LoadImagesAndLabels`. Our implementation then allows stratified splitting at a user-defined stratification level (by default 1). The splitting then allocates all substrata and images to each split such that the number of images in each split approximately follow the requested split proportions (e.g. 80-20-0% or 80-16-4%). In our case this allows us to split the data set stratified by the camera ID without rearranging the actual filestructure, if the strata are shuffled. The result is that the images in the validation split are from completely different timelapse series, than those in the training split, eliminating the data leakage problem in our project. Unfortunately it is outside the scope of this report to use this implementation to obtain k-fold crossvalidation results for our experiments, however it is easy to do using our improved data splitting functionality (not using deterministic training and repeatedly calling the run function with the same parameters will effectively do this automatically).

4 Experiments

Initially we downsample each image in the data set to have input size 640×640, such that we could train the network on our data. We performed transfer learning on the baseline model on the data set in order to be able to get an estimate of how well the baseline model could classify the phenology states of a specimen. The baseline model did not perform well enough on the data set to be useful.

4.1 Hypothesised issues and information bottlenecks

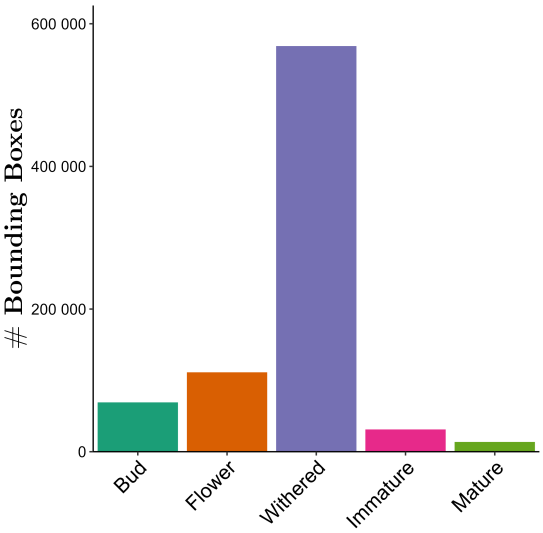
We hypothesize that the reason why the baseline model is not performing well, is partly because the bounding boxes are quite small. YOLO utilises anchors of fixed size, so when the network needs to predict several smaller objects, it will perform worse than if the objects were larger. This leads us to use some ulterior methods to counter the problem of the small bounding boxes.

The other reason is that the data set is highly imbalanced (Figure 5) causing the baseline model to only predicts objects of the **Flower** class and the **Withered** class (Table 1).

The data set has one majority class: [**Withered**] and four minority classes: [**Bud**, **Flower**, **Mature**, **Immature**].

Lastly the images contains a lot of space, that does not contribute to any learning, therefore cherry picking could improve the quality of the data. Our proposed solutions to this are threefold;

Figure 5: Class Frequency in our data set



1. Remove the **Withered** class.
2. Use Sample Weighted Loss based on a function which encapsulates the *information* in a given sample.
3. Use Inverse Class Frequency Weighted Loss.

The first option is naturally undesirable as it simple sidesteps the issue, while the latter two options are more broadly usable. We have opted not to include this option, but we have trained a model without the **Withered** class giving *very* good results ($\mathrm{mAP50\%} \approx 0.8$ and $\mathrm{mAP50\text{-}95\%} \approx 0.44$). We hypothesise that downsampling the images directly to 640×640 represents a significant information bottleneck. Our proposed solution is sliced training and inference. That is, instead of downsampling the images to the accepted input resolution of the YOLOv5s model, the images are sliced into tiles of the required size instead. This operation necessarily results in slower inference, but at the same time, the number of tiles will also be larger than the number of images, resulting in an effectively larger data set. Based on the considerations in this section we conduct the following experiments.

4.2 Slicing & Downsampling

To test the efficacy of sliced training and inference, we trained two models for 6 different degrees of downsampling, $D \in \{2,\ 2.5,\ 3,\ 3.5,\ 4,\ 4.5\}$, before slicing. The first models uses the full produced dataset, i.e. all tiles with at least one bounding box, whereas the second models use a fixed amount of tiles ($N = 2000$) which is comparable to the original number of images ($N = 2386$). This is done in order to disentangle the effect of changing tile sizes (as a fraction of the original image) from the increase in the number of samples (see Figure 3), the latter of which increases the number of batches per epoch dramatically with lower downsampling degrees resulting in an effectively slower learning rate decay.

4.3 Sample & Class Weighted Loss

To assess the effect of Sample & Inverse Class Frequency Weighted Loss, we trained the models for 25 epochs with the Sample Weighted Loss crossed with Inverse Class Frequency Loss. We trained the models on the complete sliced data set with downsampling factor $D = 4$. This was done to estimate how much we could reduce the class bias by giving increasing the weight to the minority classes and reducing the weight for the majority classes. As well as the effect of including our sample information index in the loss.

4.4 Final model

Lastly we used the results from the two experiments to train a final model (as well as a baseline model) for 200 epochs. The number 200 is chosen such that the models are trained until they are (close to) convergence. By comparing the performance of the final model to the baseline model, we will demonstrate the effectiveness of our proposed solutions to the hypothesized information bottlenecks.

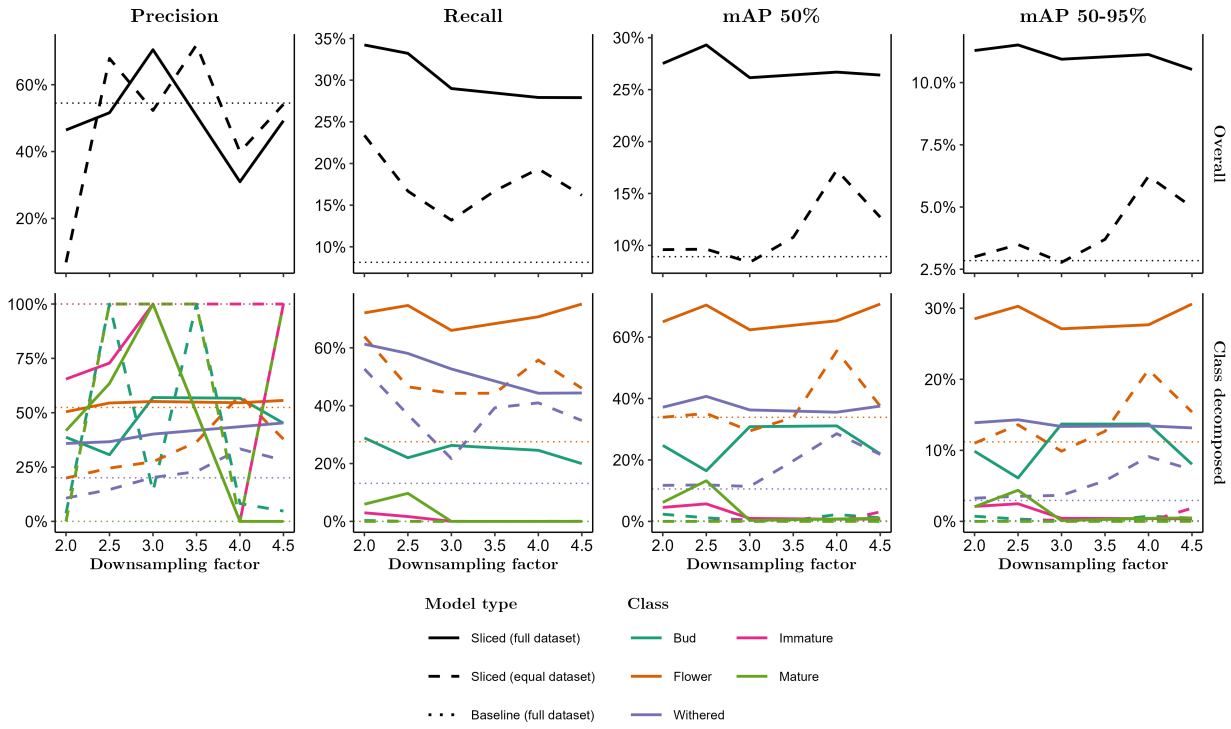
5 Results

During experimentation we initially observed that including Focal Loss might increase the performance marginally, which is why it is used by default (Table B.1). However, we ran some small experiments before the final training run, observing that Focal Loss is detrimental to the models using Class Weighted Loss (not included for brevity). For this reason the final model is trained without Focal Loss.

5.1 Slicing Experiment

After conducting the experiment and analyzing the model's performance, it is clear that the recall, $\mathrm{mAP}_{50\%}$, and $\mathrm{mAP}_{50\text{-}95\%}$ of our model improved when trained on both the full sliced dataset and the partial sliced dataset. Training on the partial sliced dataset slightly improved the model's performance, while training on the full sliced dataset led to a 30% increase in recall. The $\mathrm{mAP}_{50\%}$ and $\mathrm{mAP}_{50\text{-}95\%}$ both improved by 20% and 10%, respectively. The different models precision fluctuated greatly, indicating that they were not yet converged.

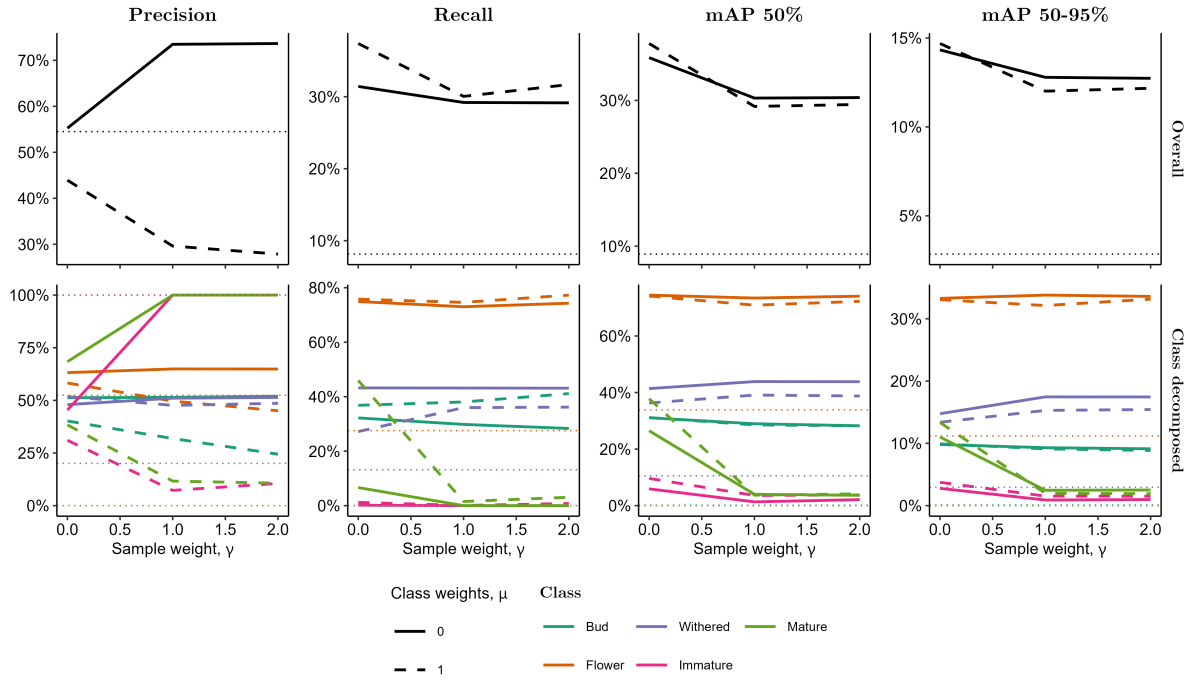
Figure 6: Model performance after 8 epochs with and without slicing across different downsampling degrees



5.2 Sample & Class Weighted Loss Experiment

After conducting the experiment with Sample & Inverse Class Frequency Weighted Loss, model improvement is not immediately obvious. While recall improved, precision decreased, however when further analyzed, we observed that the $\mathrm{mAP50\%}$ and $\mathrm{mAP_{50\text{-}95\%}}$ of the minority classes improved and the $\mathrm{mAP_{50\%}}$ and $\mathrm{mAP_{50\text{-}95\%}}$ of the majority classes decreased. This is what we expected, due to the weights being biased towards the minority classes. The experiments was partially successful, since we did not gain an overall improvement of the model, however Inverse Class Frequency Weighted Loss did result in a less biased model. So we ended up with using a sample weight $\gamma = 0$ and class weight $\mu = 1$, meaning only Inverse Class Frequency Weighted Loss was applied to achieve our final model.

Figure 7: Model performance after 25 epochs with and without class weights and using different sample weighting degrees



5.3 Final model

In the final step of our experiment, we trained a final slicing model ($D = 4$) and Inverse Class Frequency Weighted Loss for 200 epochs. We also trained a final baseline model without slicing or Weighted Loss for 200 epochs. Our results, shown in Table 2 and Figure 8, demonstrate that our proposed improvements to YOLOv5 result in significant increases in Recall (R), $\mathrm{mAP}_{50\%}$, and $\mathrm{mAP}_{50\text{-}95\%}$, with either higher or equivalent Precision (P). We observed an average increase of approximately 250% in Recall, 140% in $\mathrm{mAP}_{50\%}$, 200% in $\mathrm{mAP}_{50\text{-}95\%}$, and 13% in Precision. The performance gains were highly variable between classes, with $\mathrm{mAP}_{50\%}$ increasing by between approximately 800-1100% for the classes [**Immature**, **Mature**] and between approximately 30-100% for the classes [**Bud**, **Flower**, **Withered**]. These results are also evident in the confusion matrices of the final slicing and baseline baseline models (Figure 8), where the diagonal values for the final slicing model are consistently high and higher than the diagonal values of the final baseline model. Our results suggest that slicing significantly improves the overall performance of the model, while Inverse Class Frequency Weighted Loss reduces class bias. Sample predictions for the final slicing model are included in Figure C.1.

Table 2: Results for the final baseline and sliced models trained for 200 epochs.

Class	P	R	$\mathrm{mAP}_{50\%}$	$\mathrm{mAP}_{50\text{-}95\%}$
Baseline (Focal loss)				
Average	0.488	0.162	0.269	0.118
Bud	0.576	0.228	0.333	0.152
Flower	0.813	0.296	0.547	0.249
Withered	0.587	0.257	0.381	0.146
Immature	0.148	0.004	0.032	0.019
Mature	0.316	0.025	0.051	0.026
Baseline				
Average	0.600	0.149	0.253	0.101
Bud	0.592	0.202	0.346	0.147
Flower	0.776	0.323	0.535	0.223
Withered	0.516	0.216	0.334	0.112
Immature	0.532	0.001	0.027	0.012
Mature	0.581	0.001	0.025	0.012
Final model (Slicing)				
Average	0.683	0.549	0.596	0.328
Bud	0.686	0.477	0.528	0.265
Flower	0.786	0.651	0.725	0.373
Withered	0.763	0.535	0.662	0.384
Immature	0.609	0.434	0.451	0.250
Mature	0.570	0.649	0.615	0.369

Figure 8: Validation set confusion matrices for the final baseline and sliced models, both without focal loss.

6 Discussion

One of the main issues with the YOLO approach is that it uses fixed anchors to predict bounding boxes. If we had studied how this could have been improved more for detecting smaller objects, we might also have been able to deal with this issue differently.

In this project, we utilised the small version of YOLOv5 and trained it on images of size 640×640 for our experiments. While a larger model with greater learning capacity may have improved the performance of our network, using such a model would also increase inference time and require more RAM. Additionally, training the larger model on images of size 1280×1280 would increase the effective size of the objects being detected, but would also further increase inference time and RAM requirements. As such, we made the decision to use the smaller model and images of size 640×640 in our experiments.

Instead of using YOLOv5 as a backbone, we could also have explored different baseline models with alternative architectures to solve the object detection and bounding box prediction tasks, such as YOLOv7 or SSD. These models may have improved performance, and due to their optimisation for training, they may have allowed us to achieve convergence faster. However, we chose to use the small version of YOLOv5 in our experiments to achieve reasonable training time.

Even with these unsolved issues we were able to train a model with the use of image tiling and Inverse Class Frequency Weighted Loss weighting, that is able to detect objects of all classes with an average $\mathrm{mAP}_{50\%} = 60\%$ and $\mathrm{mAP}_{\text{50-95\%}} = 33\%$. However due to time and resource constraints, we were unable to conduct thorough hyperparameter tuning experiments in order to fine tune the hyperparameters for the final model. We could also have tried out different sample weighting indices for Sample Weighted Loss, which might also lead to a more stable training due to the unbalanced nature of our data set. YOLOv5 also utilises data augmentation, but in this project, we have not dived deeper into this. If we had managed to dive deeper into these tasks, it is very likely, that would have gotten even better results.

Our implementation of sample weighting currently uses Inverse Class Frequency Weighted Loss: $w_{n,c} = \frac{1}{\#\text{ of classes}}$, but other measures could have been used including the inverse square root class frequency or the quadratic inverse class frequency; $w_{n,c} = \frac{1}{\sqrt{\#\text{ of classes}}}$, $w_{n,c} = \left(\frac{1}{\#\text{ of classes}}\right)^2$.

Our experiments showed that using tiling on images that were downsampled by a factor of 4 significantly increased performance more than two-fold. Additionally, using Inverse Class Frequency Weighted Loss increased recall for the **Mature** class from nearly 0% to about 50%, while the **Immature** recall remained at 0% and the remaining classes' recall decreased slightly. These results were obtained from non-converged models, leading us to hypothesize that the performance gains would be greater for converged models. Our final model, which utilized both tiling and Class Weighted Loss, demonstrated a 2.5x higher $\mathrm{mAP}_{50\%}$ compared to the equivalent baseline model, with all class-individual $\mathrm{mAP}_{50\%}$ values ranging from 0.4 to 0.75. as a confirmation of this hypothesis. Since the input size of YOLOv5s is 640×640 always had a square aspect ratio, however our source images are not square but have a resolution of 6080×3420.

Downsampling images directly to the input size leads to a distortion in the width of the images, whereas using tiling does not cause distortion since downsampling is applied uniformly to both the width and height of the images. Therefore, it is difficult to determine the individual effects of maintaining the original aspect ratio and reducing downsampling on the performance of the model. These factors may be intertwined and further experimentation would be needed to tease out their individual effects on model performance. We suggest an experiment that tests the performance of sliced images as input with a downsampling factor that is based on the original image aspect ratio. This could help us identify if there exists an ever more optimal downsampling factor, which may not be uniform on width and height, that would increase the performance of the model further.

7 Conclusion

In conclusion, the use of YOLOv5 in combination with image slicing was an effective method for detecting flower phenology stages given our data set and the limited time at hand. The combination of these two techniques allowed for accurate and efficient detection of different stages of flower development, improving significantly on the performance afforded by the out-of-the-box solutions available for phenology stage detection. The use of deep learning allowed for training a powerful model, which was able to accurately solve the difficult problem of flower detection and phenology classification. Overall, this project demonstrated the potential of deep learning for visual recognition tasks in unexplored domains, and opens up the possibility for further research in this area.

References

Akyon, F. C., Onur Altinuc, S., & Temizel, A. (2022). Slicing Aided Hyper Inference and Fine-Tuning for Small Object Detection. *2022 IEEE International Conference on Image Processing (ICIP)*, 966–970. https://doi.org/10.1109/ICIP46576.2022.9897990

Cheng, Z., & Zhang, F. (2020). Flower End-to-End Detection Based on YOLOv4 Using a Mobile Device. *Wireless Communications and Mobile Computing*, *2020*, 1–9. https://doi.org/10.1155/2020/8870649

Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). YOLOX: Exceeding YOLO Series in 2021. https://doi.org/10.48550/ARXIV.2107.08430

Hiary, H., Saadeh, H., Saadeh, M., & Yaqub, M. (2018). Flower classification using deep convolutional neural networks. *IET Computer Vision*, *12*(6), 855–862. https://doi.org/https://doi.org/10.1049/iet-cvi.2017.0155

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., TaoXie, Michael, K., Fang, J., Imyhxy, Lorna, Wong, C., Yifu, Z., V, A., Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, … Xylieong. (2022). *ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations*. Zenodo. https://doi.org/10.5281/zenodo.7002879

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2014). Microsoft COCO: Common Objects in Context. http://arxiv.org/abs/1405.0312
cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C.-Y., & Berg, A. C. (2015). SSD: Single Shot MultiBox Detector. *CoRR*, *abs/1512.02325*. http://arxiv.org/abs/1512.02325

Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, *abs/1506.02640*. http://arxiv.org/abs/1506.02640

Shrivastava, I. (2020). Handling class imbalance by introducing sample weighting in the loss function. https://medium.com/gumgum-tech/handling-class-imbalance-by-introducing-sample-weighting-in-the-loss-function-3bdebd8203b4

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*.

Ween, R. E., Yoccoz, N., Høje, T. T., & Eidesen, P. B. (2022). Timing is everything: Within-plant flowering phenology impacts fruit production in the Arctic-Alpine cushion plant *Silene acaulis* (L.) Jacq. *Masters of Science in Biology - Northern Populations and Ecosystems*.

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., & Ren, D. (2019). Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. *CoRR*, *abs/1911.08287*. http://arxiv.org/abs/1911.08287

APPENDIX

A How to run the code

The source code for the project can be found in this github repository. The raw data can be downloaded from this cloud storage platform: mega folder. The raw data folder needs to be in a directory at the same level as the repository like this:

```
parent directory
├── Raw_data
└── YOLOFlower
```

Then different experiments mentioned in 4 can be done by running the **Experiments$i$.py** script, where $i \in \{0, 1, 2, 3\}$:

```
python Experiment0.py
python Experiment1.py
python Experiment2.py
python Experiment3.py
```

Predictions of images can be made by running the **detect.py** script with:

```
python detect.py --weights YOLOFlower.pt --source test_flower1.JPG
```

Predictions can be made on the following test images:

```
data
└── images
    ├── test_flower1.JPG
    ├── test_flower2.JPG
    ├── test_flower3.JPG
    └── test_flower4.JPG
```

B Hyperparameters

Table B.1: Default Hyperparameter Values

Parameter	Short name	Value
Optimizer		
Initial Learning Rate	lr0	0.01
Final OneCycleLR Learning Rate	lrf	0.01
Momentum or β_1	momentum	0.937
Weight Decay	weight_decay	0.0005
Warmup epochs	warmup_epochs	2.0
Warmup Momentum	warmup_momentum	0.6
Warmup Bias Learning Rate	warmup_bias_lr	0.1
Loss Components		
Box Loss Multiplier	box	0.15
Class Loss Multiplier	cls	0.3
Class Loss Positive Weight	cls_pw	1.0
Object Loss Multiplier	obj	0.7
Object Loss Positive Weight	obj_pw	1.0
Bounding boxes and Anchors		
Intersection over Union Threshold	iout_t	0.2
Multiple Anchor Threshold	anchor_t	3.0
Anchors per Output Layer	anchors	10.0
Focal Loss		
Focal Loss Gamma	fl_gamma	0.0
Augmentation		
HSV-Hue (%)	hsv_h	0.015
HSV-Saturation (%)	hsv_s	0.1
HSV-Value (%)	hsv_v	0.1
Image Rotation (°)	degrees	0.0
Image Translation (%)	translate	0.1
Image Rescale (gain)	scale	0.1
Image Shear (°)	shear	0.0
Image Perspective (%)	perspective	0.0
Image Vertical Flip (%)	flipud	0.5
Image Horizontal Flip (%)	fliplr	0.5
Image Mosaic (%)	mosaic	0.5
Image Mixup (%)	mixup	0.0
Segment Copy-Paste (%)	copy_paste	0.0

C Example prediction

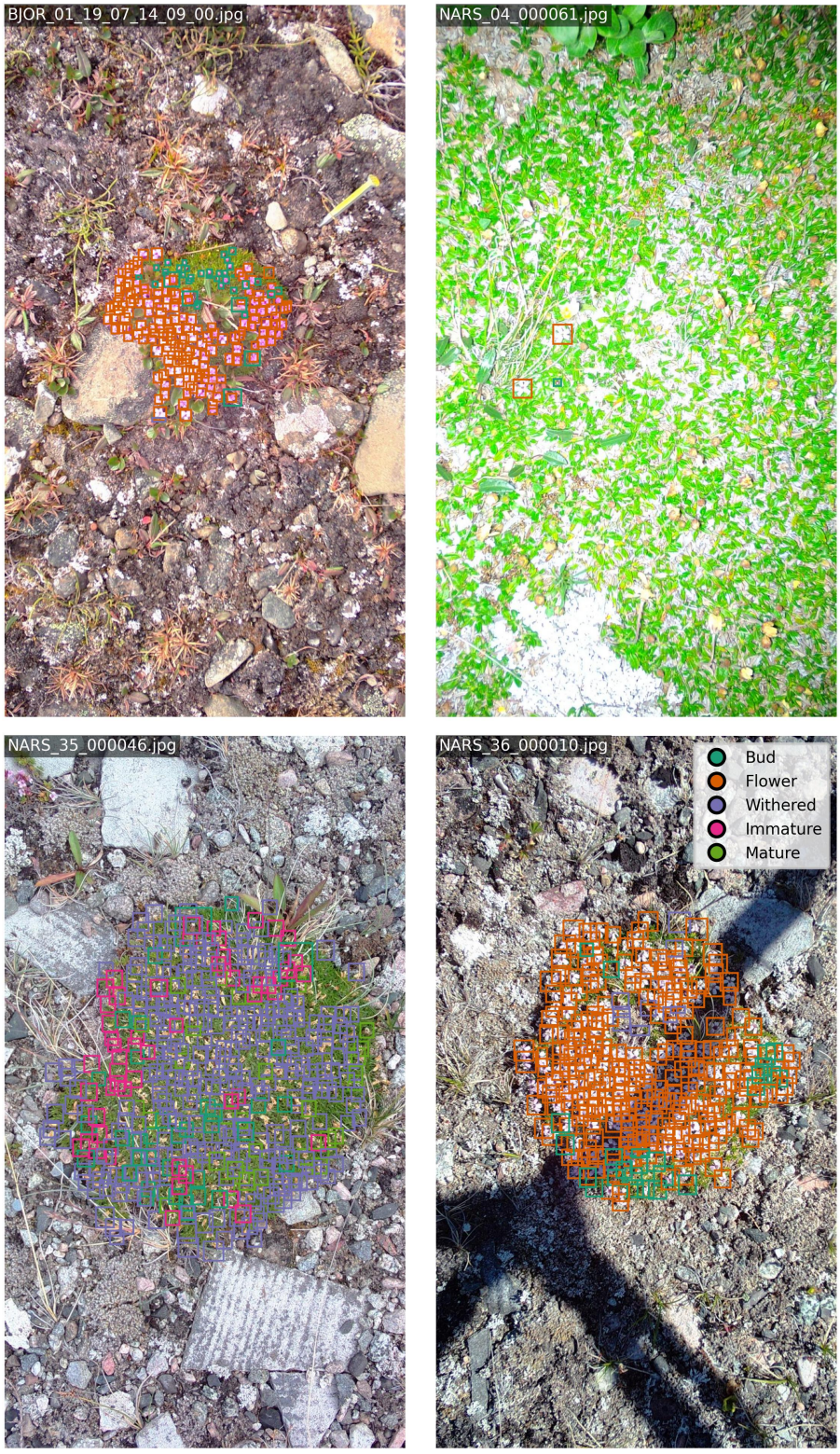


Figure C.1: Example of predictions using our best model for 4 random images in the data set.

D Encoder architecture

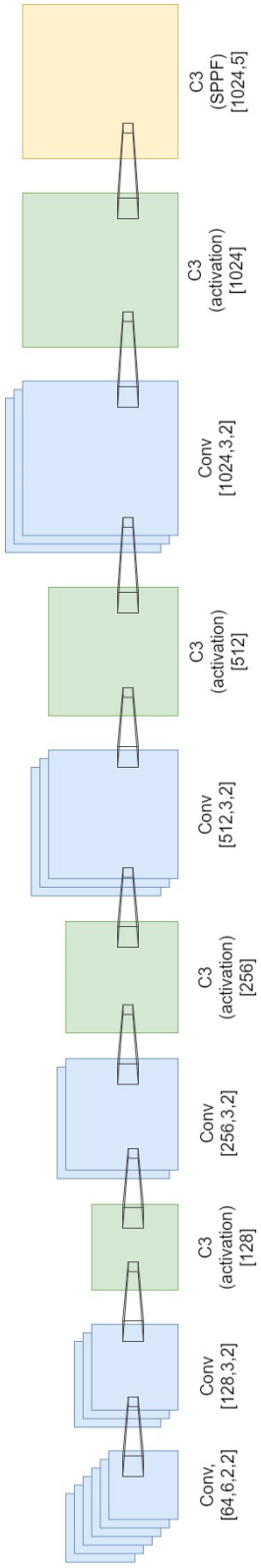


Figure D.1: Architecture of the Backbone or Encoder