Accessible deep learning for camera trap data analysis

Final report of Research Experience Placement

Bartosz Dzionek

6th September 2021

Abstract

Camera trap surveys generate millions of images that are manually tagged by scientists. Machine learning classifiers can significantly speed up this process, but they are hard to use for zoologists without technical skills. *EasyCT* is a web application that aims to be accessible for people without coding expertise. It can be run locally on any computer and read images stored on external hard drives. The application provides insight into the distribution of photos over time and implements a single species classifier. The classifier was found to give high accuracy classification on badgers. At the same time, it is quick and can process four high-resolution photos per second on a mid-level computer. *EasyCT* shows the potential to reduce the time-consuming process of tagging camera trap images. It is open-source and can be extended by the community with different machine learning models.

1 Introduction

1.1 Context and motivation

Camera traps are valuable for biodiversity monitoring as they are scalable and well-suited to solve global challenges in zoology [1]. They can generate millions of photos that need to be later analysed. Within camera trap data analysis, identifying species is the most crucial and time-consuming part [2]. As a result, to handle the data volume, studies may need to reduce the number of cameras, geographical coverage and their duration [3]. It was found that deep learning models can significantly reduce the time necessary to identify species with high accuracy [4]. However, those models face challenges with transferability and do not work well on habitats different to their training set [5]. What is more, they require advanced computer skills that many zoologists do not possess [5].

1.2 Previous work

Various machine learning algorithms have been developed for image recognition. Probably, the most well-known zoological solution is *MegaDetector*, created by *Microsoft* [6]. However, it is intended for species detection rather than classification. It has its unofficial graphical user interface developed by Petar Gyurov in *Electron.JS* [7]. Another alternative interface for *Megadetector* is being developed by Naomi McWilliam at ZSL [8].

Apart from *Megadetector*, there are some scripts and applications designed specifically for classification. There is *Camera Trap Image Classifier* created in Python [9] and *MLWIC2* written in R Shiny [5].

1.3 Objectives

The main objective of the research internship was to create a tool that can leverage deep learning algorithms for single species classification. The software needed to be simple to install and use. Moreover,

other researchers working in zoology should be able to extend the software in the future. To make that possible, the programming language of the application should be familiar to the community. Hence, I picked R as it is the most widely used statistical programming language in academia [10].

The secondary goal was to evaluate the software in the context of badgers in Cornwall. I explain this data set in more detail in the *Data* section. To make the application worth using, the accuracy of classification needed to be high.

2 Software

2.1 Availability

The whole code of *EasyCT* is available through a GitHub repository:

`https://github.com/dzionek/easyCT`

The software is open-source and licensed under the MIT license described inside the repository.

2.2 Simple installation

I have written the application in the *R Shiny* web framework. Besides the code in R, some parts of it require Python, Perl and C++. I have used *Exiftool* to extract metadata from images. Since one of the goals was to make the application easy to install, I designed the software to run from a Docker container.

I thoroughly described the exact installation process inside the *README* file of the repository. It requires only to download Docker and execute two commands in a terminal. The commands pull the official image from *Docker Hub*, mount local path within the container and run it. The installation process is as hard as its hardest part. In this case, the most challenging part is installing Docker. But once this is done, the user is guaranteed to use the same version of dependencies as the developer.

The Docker image currently supports x86-64 architecture on Windows, Linux and Mac. There are plans to create a separate image for ARM64 that will cover Apple Silicon processors. The current alternative for ARM64 users is downloading the dependencies themselves by following the *Dockerfile* in the repository.

2.3 Classification models

The classifier inside the app is an adapted version of a classifier created by ZSL and Imperial College London. It is based on *Inception-v3* and will be described in Li et al (article in preparation). The classifier is a convolutional neural network without a fully connected top layer, with maximum pooling and *Imagenet* weights. Its input image format is $299x299$.

The classifier can be used for single species classification. The user needs to provide a positive and a negative class to train the algorithm and create their models. The provided sets will split randomly into a train and test set in a ratio of 75% to 25%.

2.4 Graphical User Interface

The application opens in any modern web browser. It has a form of a dashboard with three tabs: Description, Time and Classification. The first tab explains how to use the app.

The Time tab uses Exif metadata to extract the time of image creation. After selecting a directory of photos, the user can see a histogram of images over time. Another panel shows a histogram of photos with regard to days of the week or hours of the day. There is also a file selector used to choose the photos taken within a given time range.

The Classifier tab is used for single species classification. Users can build their own supervised models using the classifier or use ones already created. If they want to create their own model, they need to select positive and negative class directories. After training a model, the application displays the

model's evaluation metrics, including a confusion matrix. Once the model is selected, users can classify their unlabelled photos. Before doing this, the user can select the number of pixels that should be trimmed from the top or bottom of each image. The purpose of this is to remove info bars from photos. Otherwise, a particular part of the bar (e.g. temperature, time) might affect classification. The classification result is a CSV file with the predicted class and the probability that a given photo belongs to the positive class. This file can be used for further analysis in simple tools such as *Microsoft Excel* spreadsheets. More advanced users can use it inside a programming language environment.

3 Analysis

This section shows an application of *EasyCT* for analysing badgers in Cornwall. The goals are to find any patterns in the histograms and to check the accuracy of the classifier.

3.1 Data

This section describes the set of camera trap photos collected by ZSL's researchers. This data set is used as an example application of *EasyCT*. However, the application can handle any type of camera trap photos. It can classify any images, not only those obtained in camera trap studies.

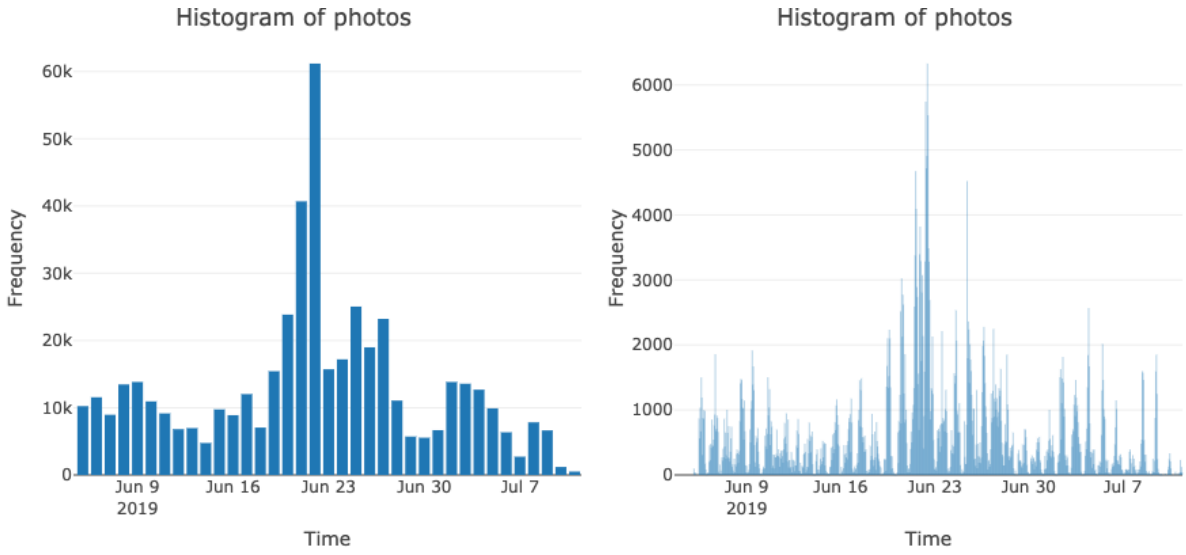
Data origin The cameras were deployed in 86 locations in forests and fields of Cornwall, United Kingdom. The time frame of the photos ranges from 5th June to 11th July 2019.

Data description The data set comprises 480 thousand photos of a total size of 757 gigabytes. A sample photo is shown in Figure 1. Every image of the data set contains an info bar at the bottom. The bar is 38 pixels high and includes the local temperature, time and ID. In the bottom-left corner, there is a watermark of the camera manufacturer (*Browning Trail Cameras*).

Figure 1: A sample photo from the data set capturing a pheasant.

3.2 Time analysis

The Time tab can be used to find patterns in the data set. Loading the whole data set took around 2 hours 10 minutes (computer specification in appendix). The histograms of photos are shown in Figure 2.



(a) daily bars

(b) hourly bars

Figure 2: Histograms of photos over time

The daily histogram (Fig. 2a) has its highest value on 22nd June 2019. Analysing this particular day in more detail in the hourly histogram (Fig. 2b) shows this spike is especially evident between 11:00 and 12:00. If we put this specific date and hours in the photo selector, many of the photos were captured by one camera. We can focus on this one camera and filter entries stored in its directory (Fig. 3). Then, the selector will show

```
3,873 entries (filtered from 6,330 total entries).
```

That is about 61% of the data collected at this hour. If we pick a random photo from the ones selected, we will realise the problem is caused by vegetation moving in front of the camera (Fig. 4). Hence, we can remove the photos recorded by the camera within these hours and save time needed to manually inspect 3873 images.

We can also check how the photos distribute over a week or day. We can see the number of photos distributes almost uniformly over days of the week (Fig. 5a). Only Sunday has a much lower number than the other days. On the other hand, the distribution over 24 hours of the day shows a big difference between days and nights (Fig. 5b). The highest number of photos is around noon and is much lower at night. That may be caused by the animal sleep patterns and reduced number of false triggers due to the Sun inactivity.

Selected photos

Show 10 entries Search: /root/photos/C4 camera ti

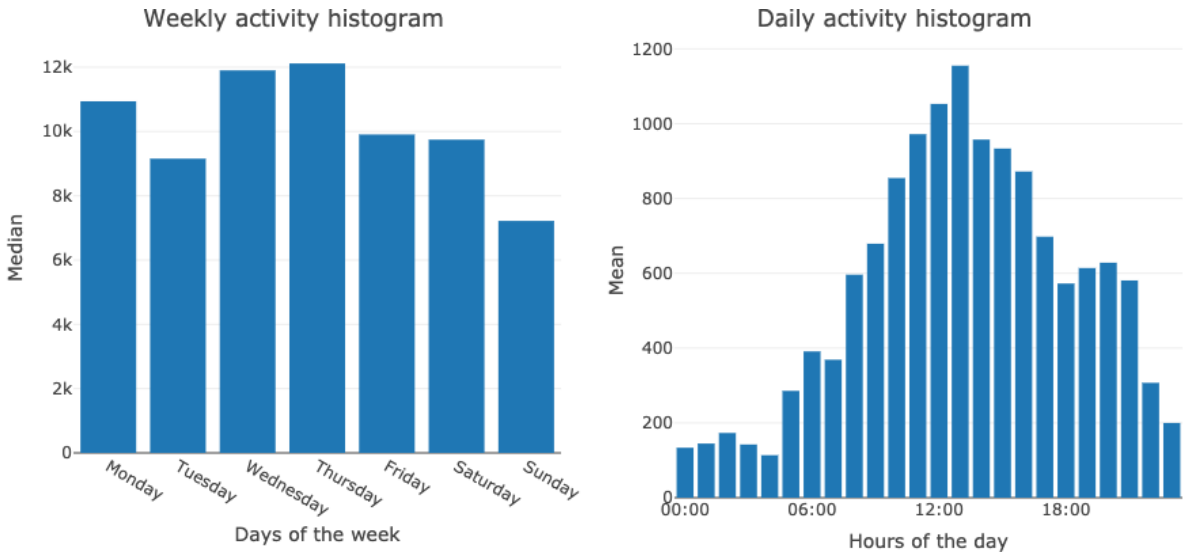
	SourceFile	datetime
1	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6206.JPG	2019-06-22T11:00:00Z
3	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6207.JPG	2019-06-22T11:00:01Z
4	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6208.JPG	2019-06-22T11:00:02Z
5	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6209.JPG	2019-06-22T11:00:05Z
7	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6210.JPG	2019-06-22T11:00:06Z
9	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6211.JPG	2019-06-22T11:00:07Z
10	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6212.JPG	2019-06-22T11:00:07Z
12	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6213.JPG	2019-06-22T11:00:08Z
13	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6214.JPG	2019-06-22T11:00:08Z
15	/root/photos/C4 camera traps/P18/DCIM/101_BTCF/IMG_6215.JPG	2019-06-22T11:00:09Z

Showing 1 to 10 of 3,873 entries (filtered from 6,330 total entries)

Previous 1 2 3 4 5 … 388 Next

Figure 3: Photos filtered by the photo selector.

Figure 4: Problematic vegetation.



(a) Median number of photos by day of the week

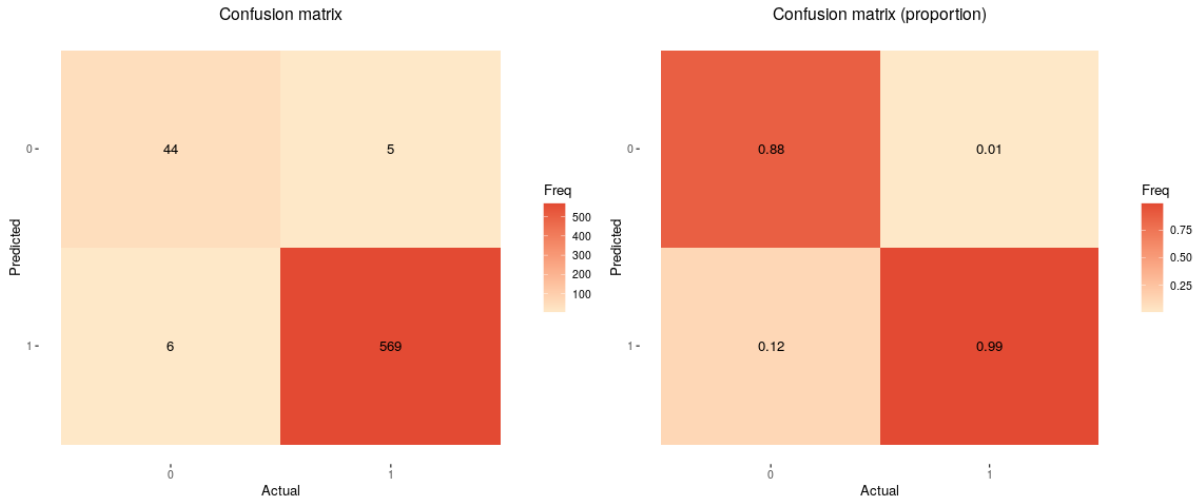
(b) Mean number of photos by hour

Figure 5: Histograms averaged over week and day

3.3 Classifying badgers

In this section, I have created a model using the application's classifier. While doing this, I checked how well the classifier could adapt to photos from different camera trap surveys. In particular, I trained a model using the ZSL data set. Then, I classified badger photos collected in a study of the University of Sheffield [11].

To train the model, I used a sample of photos manually classified by ZSL's scientists. The input set comprised 2295 photos of badgers and 200 random photos without badgers. Because I used the model on images from a different survey, I needed to trim both the info bar and the watermark (96 pixels from the bottom in total). Building the model took 16 minutes and yielded a model of the accuracy of 0.982 and loss of 0.252. The confusion matrix is shown in Figure 6.

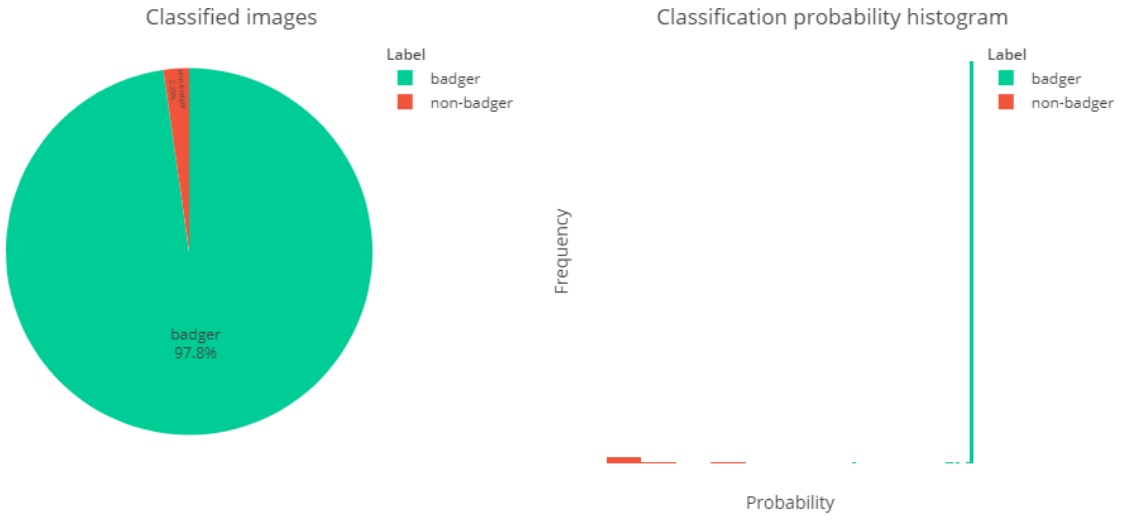


(a) Number of photos format

(b) Proportion format

Figure 6: Confusion matrix of the compiled model.

The model name is `badgers_zsl` and can be downloaded from the GitHub repository of this report. Then, the model was used to classify 1556 photos of badgers from the University of Sheffield. These photos had an info bar of 31 pixels that was trimmed with the app. I left the default threshold of 0.5. After 6 minutes, the results were as described by the graphs in Figure 7.



(a) Proportion of predicted classes

(b) Histogram of probability a given photo is positive

Figure 7: Results of the classification.

The classification of badgers was accurate in 97.8% of cases (Fig. 7a). However, the histogram (Fig. 7b) showed 22 photos with a probability of almost zero). When looking more closely into the CSV file generated by the classifier, I found 9 photos with the lowest probability (Fig. 8). Even though the accuracy is relatively acceptable, the model struggled with photos in which only a tiny part of a badger was visible; a badger was blurred or similar to the photo background.

Figure 8: The nine misclassified photos of the lowest probability.

4 Discussion and conclusions

4.1 Summary

The application proved to be useful for camera trap data analysis. It is easy to use at the cost of limited functionality.

The time tab can quickly summarise the data set and produce publication-ready plots. However, it can only extract the `DateTimeOriginal` tag from the Exif data. If a particular photo is malformatted and does not have this tag, this tab might be of less help.

On the other hand, the classifier tab allows users to build relatively accurate models with a reasonably low number of photos. Moreover, model training and classification is quick and shows the potential

to accelerate the mundane process of photo tagging. The classifier accuracy is not ideal. But it can be improved by comparing results from different models and using more comprehensive training sets.

4.2 Comparison with any other related work

The application is simpler to install and use than *Megadetector* and *MLWIC2*. Both of these two require the user to go through many steps in the documentation and install dependencies on their own [5] [6]. Nonetheless, *Camera Trap Image Classifier* is an app that was also containerised and can be pulled from Docker Hub. But its documentation is not very specific about Docker and may be hard to follow for people new to Docker [9].

It is worth mentioning that the considered alternatives offer much more sophisticated algorithms that can handle multiple species, perform data augmentation and transfer learning. It is expected that the classifier used in *easyCT* can be quicker due to its simplicity. More research might be needed to compare the classifiers and resolve the trade-off between simplicity, time effectiveness and functionality.

4.3 Improvements and extensions

The application allows the user to train a model and classify photos. The final result of the classification is a CSV file. Any further analysis needs to be conducted beyond the app. It might be beneficial to extend the current interface to manipulate photos after they were classified. One possibility is to move the images to two different directories based on their predicted class. Another option is to move photos to n directories based on n probability intervals

$$\left[0, \frac{1}{n}\right), \left[\frac{1}{n}, \frac{2}{n}\right), \ldots, \left(\frac{n-1}{n}, n\right].$$

That would allow scientists to quickly check for false positives and negatives, or choose the appropriate threshold. However, it is vital to maintain the original configuration, i.e. information about a particular site and survey of any photo. Therefore, it might be better to visualise images and their predicted values without changing the original directory structure.

What is more, the application can be adapted to work faster than currently. The first possibility is to activate GPU support on the Docker container. Utilising the power of GPU will give a significant improvement for powerful computers. Though, it might not change anything on lower-end laptops. It should also be possible to parallelise the extraction of features for the model.

Another possibility to improve the application is to use more models. Currently, the only supported model is the version of the *Inception-v3* model described earlier. Nonetheless, it is not a guarantee that this is the best model. It would be beneficial if scientists can select different models and compare their results. Or, the app can automatically select the best model based on the evaluation metrics of all models on the train and test set.

Moreover, in its current version, the app can only do single species classification. If someone wants to classify multiple species, they might need to repeat the single species classification many times. It is possible to simplify this by constructing a multi-species classifier.

5 Acknowledgements

I would like to thank Dr Marcus Rowcliffe (ZSL) for supervising my internship, introducing me to the problem described in this paper and testing the software. I also thank Verity Miles (Imperial College London) for mentoring, testing the software and proofreading this paper. My work was supported by the Natural Environment Research Council's scheme of Research Experience Placement and The Grantham Institute at Imperial College London.

References

[1] Robin Steenweg, Mark Hebblewhite, Roland Kays, Jorge Ahumada, Jason T Fisher, Cole Burton, Susan E Townsend, Chris Carbone, J Marcus Rowcliffe, Jesse Whittington, Jedediah Brodie, J Andrew Royle, Adam Switalski, Anthony P Clevenger, Nicole Heim, and Lindsey N Rich. Scaling-up camera traps: monitoring the planet's biodiversity with networks of remote sensors. *Frontiers in Ecology and the Environment*, 15(1):26–34, 2017.

[2] Jürgen Niedballa, Rahel Sollmann, Alexandre Courtiol, and Andreas Wilting. camtrapr: an r package for efficient camera trap data management. *Methods in Ecology and Evolution*, 7(12):1457–1462, 2016.

[3] Michael A. Tabak, Mohammad S. Norouzzadeh, David W. Wolfson, Steven J. Sweeney, Kurt C. Vercauteren, Nathan P. Snow, Joseph M. Halseth, Paul A. Di Salvo, Jesse S. Lewis, Michael D. White, Ben Teton, James C. Beasley, Peter E. Schlichting, Raoul K. Boughton, Bethany Wight, Eric S. Newkirk, Jacob S. Ivan, Eric A. Odell, Ryan K. Brook, Paul M. Lukacs, Anna K. Moeller, Elizabeth G. Mandeville, Jeff Clune, and Ryan S. Miller. Machine learning to classify animal species in camera trap images: Applications in ecology. *Methods in Ecology and Evolution*, 10(4):585–590, 2019.

[4] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S. Palmer, Craig Packer, and Jeff Clune. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences*, 115(25):E5716–E5725, 2018.

[5] Michael A. Tabak, Mohammad S. Norouzzadeh, David W. Wolfson, Erica J. Newton, Raoul K. Boughton, Jacob S. Ivan, Eric A. Odell, Eric S. Newkirk, Reesa Y. Conrey, Jennifer Stenglein, Fabiola Iannarilli, John Erb, Ryan K. Brook, Amy J. Davis, Jesse Lewis, Daniel P. Walsh, James C. Beasley, Kurt C. VerCauteren, Jeff Clune, and Ryan S. Miller. Improving the accessibility and transferability of machine learning algorithms for identification of animals in camera trap images: Mlwic2. *Ecology and Evolution*, 10(19):10374–10383, 2020.

[6] Microsoft Research. Megadetector, 2020. `https://github.com/microsoft/CameraTraps/blob/master/megadetector.md`. Retrieved on 26th August 2021.

[7] Petar Gyurov and Alexandar Gyurov. Megadetector gui, 2021. `https://github.com/petargyurov/megadetector-gui`. Retrieved on 27th August 2021.

[8] Naomi McWilliam. Megadetector interface, 2021. `https://github.com/NaomiMcWilliam/Megadetector-Interface`. Retrieved on 27th August 2021.

[9] Marco Willi, Ross T. Pitman, Anabelle W. Cardoso, Christina Locke, Alexandra Swanson, Amy Boyer, Marten Veldthuis, and Lucy Fortson. Identifying animal species in camera trap images using deep learning and citizen science. *Methods in Ecology and Evolution*, 10(1):80–91, 2019.

[10] Chelsea Loomis Lofland and Rebecca Ottesen. The sas versus r debate in industry and academia. In *SAS Global Forum 2013. SAS*, pages 348–2013. Citeseer, 2013.

[11] Lyudmila Mihaylova, Ruth Little, Ruilong Chen;, Richard Delahay, and Ruth Cox. Badger datasets for image recognition. 6 2019.

A Appendix: Benchmark specification

All benchmark times were measured on the following configuration:

Type: Desktop PC **CPU:** *Intel i5-6500* **GPU:** *GeForce GTX 1060 6GB*
RAM: 16GB **System:** *Windows 10 Pro* **Docker Engine:** v20.10.7 Hyper-V backend