




Sherlock—A flexible, low-resource tool for processing camera-trapping images

Matthew J. Penn¹  | Verity Miles² | Kelly L. Astley² | Cally Ham²  |
Rosie Woodroffe²  | Marcus Rowcliffe² | Christl A. Donnelly^{1,3}

¹Department of Statistics, University of Oxford, Oxford, UK

²Institute of Zoology, London, UK

³Pandemic Sciences Institute, University of Oxford, Oxford, UK

Correspondence

Matthew J. Penn

Email: matthew.penn@st-annes.ox.ac.uk

Funding information

Cornwall Wildlife Trust; Engineering and Physical Sciences Research Council; Natural Environment Research Council, Grant/Award Number: NE/L002515/1 and NE/S007415/

Handling Editor: Diana Fisher

Abstract

1. The use of camera traps to study wildlife has increased markedly in the last two decades. Camera surveys typically produce large data sets which require processing to isolate images containing the species of interest. This is time consuming and costly, particularly if there are many empty images that can result from false triggers. Computer vision technology can assist with data processing, but existing artificial intelligence algorithms are limited by the requirement of a training data set, which itself can be challenging to acquire. Furthermore, deep-learning methods often require powerful hardware and proficient coding skills.
2. We present Sherlock, a novel algorithm that can reduce the time required to process camera trap data by removing a large number of unwanted images. The code is adaptable, simple to use and requires minimal processing power.
3. We tested Sherlock on 240,596 camera trap images collected from 46 cameras placed in a range of habitats on farms in Cornwall, United Kingdom, and set the parameters to find European badgers (*Meles meles*). The algorithm correctly classified 91.9% of badger images and removed 49.3% of the unwanted 'empty' images. When testing model parameters, we found that faster processing times were achieved by reducing both the number of sampled pixels and 'bouncing' attempts (the number of paths explored to identify a disturbance), with minimal implications for model sensitivity and specificity. When Sherlock was tested on two sites which contained no livestock in their images, its performance greatly improved and it removed 92.3% of the empty images.
4. Although further refinements may improve its performance, Sherlock is currently an accessible, simple and useful tool for processing camera trap data.

KEYWORDS

camera-trapping, image classification

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2023 The Authors. *Methods in Ecology and Evolution* published by John Wiley & Sons Ltd on behalf of British Ecological Society.

1 | INTRODUCTION

Camera-trapping methods have the potential to provide profound insights into the behaviour and ecology of animal populations (Kucera & Barrett, 2011) and are becoming increasingly widespread across the world (Cole Burton et al., 2015). Development in technology has enabled the simultaneous study of a range of species across large spatial scales and with minimal disturbance. It is a substantially more cost-effective method of surveying mammals than the more traditional live trapping (De Bondi et al., 2010) and has led to the creation of extensive digital data sets of images (Green et al., 2020) on a scale that was previously unachievable.

A common constraint of camera-trapping research is the vast quantity of data produced. Millions of images can be generated from a single survey, a large proportion of which may be 'empty' (i.e. not containing any animals), as a result of moving vegetation and changes in light levels which falsely trigger the infrared sensors. This is particularly evident where camera locations are randomly selected and not designed to maximise encounters with the target species (Marcus Rowcliffe et al., 2008).

To help analyse such data, a number of software packages have been developed by researchers across the world to automatically classify images from camera traps (Meek et al., 2020; Microsoft, 2022; Schneider et al., 2018; Tabak et al., 2020). These have proved successful, both at simply removing empty images (Wei et al., 2020) and at identifying animal species (Villa et al., 2017) and can achieve similar levels of accuracy to classification by human experts (Norouzzadeh et al., 2018).

Despite this process, many researchers face limited options when choosing a software package. A number of packages require at least some coding background to install and use (Green et al., 2020), and many require expensive hardware such as GPUs (graphics processing units) to run (Wei et al., 2020). Moreover, with the majority of new algorithms being based on deep-learning methods (Meek et al., 2020; Schneider et al., 2018; Tabak et al., 2020), researchers are forced to find one that has already been trained on appropriate data, or obtain a large amount of manually classified images, done by either the researchers themselves (Schneider et al., 2020) or by the public (Adam et al., 2021).

This issue has recently been partly addressed by the publication of GUI (graphical user interfaces) for the 'MegaDetector' algorithm (Beery et al., 2019; van Lunteren, 2023). The vast amounts of training data used in the construction of this algorithm mean that it is transferable to a wide range of habitats, while it is possible to run it on a CPU. However, this comes at a computational cost—in our experiments, it ran slowly, taking 7.04 s per image on a standard desktop computer (compared to Sherlock's 0.98 s per image), making it unsuitable as a stand-alone for projects with large amounts of data.

The goal of this study was to provide an alternative solution to these issues through the development of a flexible and, crucially, low-resource algorithm, Sherlock, that is accessible to researchers with little or no computing background. This algorithm attempts to identify empty images—that is, images that were triggered by

an inconsequential movement, such as a plant blowing in the wind. The final product is available on Zenodo (Miles, 2023) and contains simple, detailed instructions on how to use it, including a guide to installing Python.

Sherlock does not require any training data but instead uses an intuitive approach. It examines the sequence of images and looks for marked deviations from a rolling background image, allowing it to remove a large proportion of the false positives from the data. It also does not require large amounts of computing power or a GPU—indeed, it runs off a single core—meaning its hardware requirements are minimal, and that the computer it is running on can continue to be used for other tasks. Moreover, it can be easily paused and restarted, meaning that there is no requirement for it to be continuously run until completion.

Other authors have sought to produce algorithms that do not require any training data, such as Xi et al. (2021). The most similar algorithm that the authors have found to Sherlock is the Zilong algorithm given in Wei et al. (2020). Zilong is founded on similar principles—it does not use neural networks but instead seeks to compare images to the background and is shown to perform well on a range of images in Wei et al. (2020). However, Zilong, alongside the algorithm presented in Xi et al. (2021), requires cameras to take 'bursts' of images—that is, a single movement triggers a sequence (of at least a certain, fixed length) of images to be taken. However, this assumption was not met in the data on which our algorithm was tested as many encounters between animals and cameras would cause only a single image to be recorded. Moreover, Zilong is more difficult to set up than Sherlock, and so we believe we offer a valuable alternative to a wide range of researchers.

This paper is structured as follows. In Section 1, we give an explanation of the method behind Sherlock, illustrated with an example from our data set, and also detail the ways in which it can be parameterised. In Section 2, we present the results of testing Sherlock on a set of camera traps positioned across Cornwall, in the south-west of England. Finally, in Section 3, we discuss these results and detail future improvements that could be made to the algorithm.

2 | MATERIALS AND METHODS

2.1 | Test data

We tested the performance of Sherlock against a data set of camera trap images that had been separately classified by one of the authors (VM). Images were obtained from two camera surveys conducted in 2019 in Cornwall, United Kingdom, as part of a study investigating analytical methods of estimating badger density. We used a random sample of 23 camera deployments from each site, totalling 240,596 images. Images were classified by a human as either containing a badger or not and were tagged using (XnView MP, 2020). The data set contained 306 badger images, with the vast majority being non-badger images but potentially containing other moving subjects such as wildlife, livestock, people and

vehicles. The cameras were placed randomly within the survey area rather than being placed strategically to maximise encounters with the target species, in order to satisfy the assumptions of the 'random encounter' model for estimating population density (Marcus Rowcliffe et al., 2014). This means that some of the sites had a large number of disturbances caused by sheep and cattle. We also tested performance with varying model parameters using a smaller data set of 1000 images from a single camera deployment, including 68 classified badger images.

A selection of images which Sherlock correctly tagged as containing badgers is shown in Figure 1, illustrating the range of habitats in which it was used.

2.2 | Sherlock code

In this section, the method of the code is detailed, alongside an example. This example is taken from a set of 16 images, shown in Figure 2 which were the only images captured by this camera on the night that they were taken, and hence would have been analysed as a single set by the code (see below for details). The image on which the majority of the analysis is illustrated is image (i).

2.2.1 | Rolling background image

The main way in which animals are identified is by comparing each image to a rolling background image. Using a rolling background, rather than a fixed one for the whole set of images is particularly helpful for camera traps that are left for a long period of time, where vegetation growth or other factors may change the background. In an ideal setting, this background image would contain the closest N images (for some N) to the current image. However, this would require computing a separate background image for each image—a computationally expensive task—and instead the background image is periodically updated.

There are two conditions which determine the set of images that comprise each background image. Firstly, there is a user-inputted maximum number of images. This is important as if too many images are used then the background may shift as, for example, light conditions change. Moreover, using a very large number of images may fill up the memory of the computer used and cause the algorithm to crash. The second condition is that the background can only be formed from a set of consecutive daytime images or a set of consecutive nighttime images. Each image is identified as either 'day' or 'night' by accessing the metadata of the image to determine whether the flash was used.



FIGURE 1 A selection of images which Sherlock correctly identified as containing badgers.

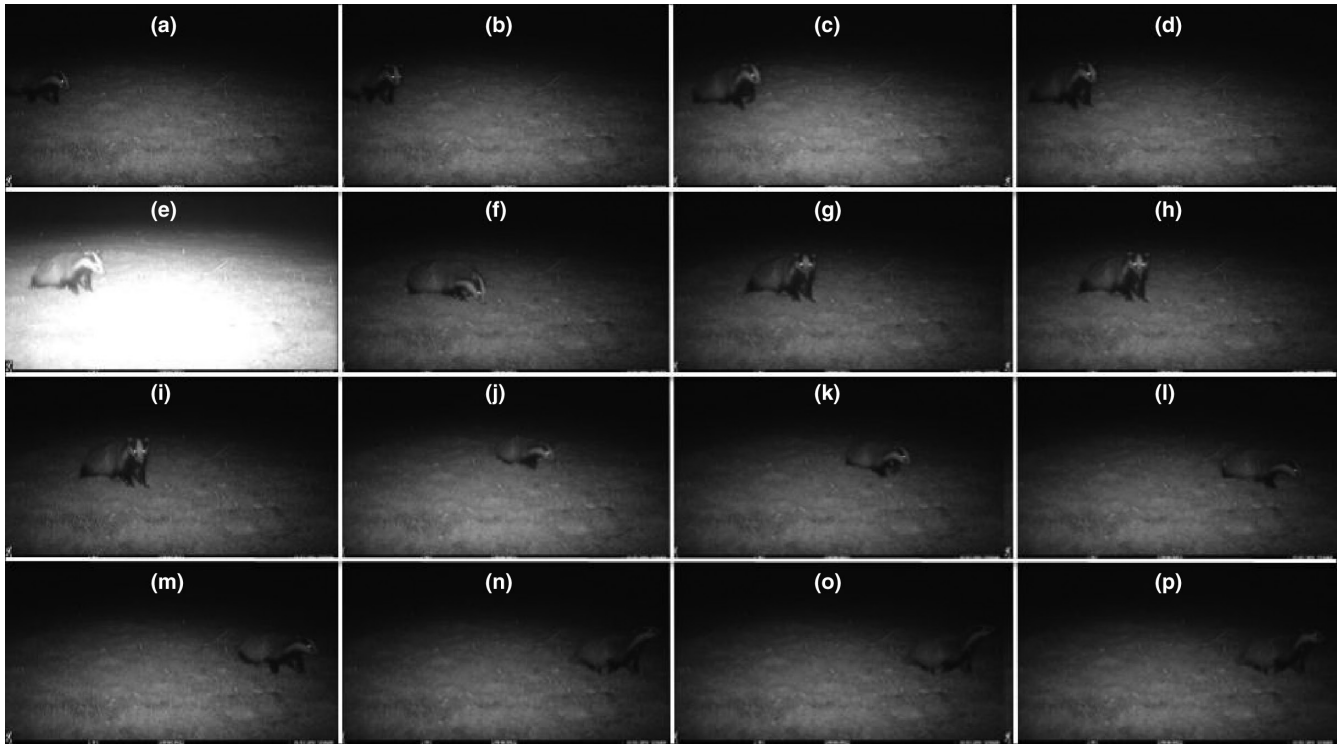


FIGURE 2 The original set of 16 images which were used to make this example. These images, labelled from (a) to (p), show a badger moving across the field of view of the camera. Although the camera's infrared flash is automatically adjusted to ensure correct exposure, image (e) is evidently overexposed. Image (i) was chosen to illustrate the image analysis procedure.



FIGURE 3 The background image created from the 16 images given in [Figure 2](#).

Once the set of images has been chosen by the algorithm, the background image is then formed by taking the median colour of each pixel in this set. Using the median instead of the mean prevents large deviations from being overweighted, giving a more accurate construction of the true background. More advanced models have been developed, such as Gaussian mixture modelling (Hari Hara Santosh et al., 2013). However, the use of these would incur a substantial additional computational cost, and so the median was chosen as a compromise between accuracy and efficiency.

An example of a background image, created from 16 images, is given in [Figure 3](#), where one can see that the background is recovered, despite the badger walking across the foreground, and the

image where the flash caused the foreground to be much lighter. However, the background is not too blurry, as it retains some details such as individual blades of grass, which is an advantage of this background being made from a reasonably small number of images.

2.2.2 | Bounding box creation

Once the background image has been calculated, it is possible to test each image and to determine whether it contains an animal. This is done by randomly sampling pixels, testing them and then attempting to build up bounding boxes of 'disturbances'—connected sets of

FIGURE 4 A graphic showing the result of sampling 5000 random pixels image I in Figure 2. Green circles indicate that a pixel was not counted as a disturbance, while red pixels indicate that a pixel was counted as a disturbance. Note that the circles have been set to have a radius of 3 pixels for demonstrative purposes although only a single pixel was analysed in each case.



pixels that are different from the background image and meet any other criteria (detailed below).

The initial sample of pixels is chosen uniformly (i.e. every pixel has an equal chance of being in the sample) at random from the image. The number of pixels sampled can be modified by the user. To determine whether this pixel is a 'disturbance', it must satisfy the following conditions:

1. It must be sufficiently different from the equivalent pixel in the background image (i.e. the maximum difference between the RGB values (red, green, blue; with each value being between 0 and 255) of these two pixels must exceed some threshold, which can be different for day and night images)
2. If appropriate, the RGB colours of the pixel must fall between a user-specified upper and lower bound. This condition can be removed by setting the lower bound to (-1, -1, -1) and the upper bound to (256, 256, 256), and was not used by the algorithm when looking for badgers. However, if the animal in question was (at least largely) a single colour, then this parameter could be set so that only pixels close to that colour were considered.
3. If appropriate, the pixel in the image must be sufficiently grey (i.e. the R, G and B values of the pixel must have a range smaller than some threshold). This condition can be removed by setting the threshold to be 256, as in many cases (when the species of interest is not approximately greyscale), it will not be appropriate.

Figure 4 shows the results of carrying out this sampling procedure on an image. Most of the disturbances found were indeed on the body of a badger, although there is a minority of small clusters that were caused by moving vegetation or changing light conditions. It is also worth highlighting the number of pixels on the badger that were *not* counted as disturbances, and this noisy feature means that the image requires subsequent analysis.

The area around each pixel that is found to be a disturbance is then analysed further, through what we call the 'bouncing' procedure. The algorithm attempts to find paths of pixels moving outward from the disturbance, as illustrated in Figure 5. It tries four initial directions (upwards, downwards, leftwards and rightwards) in turn, continuing the

path in that direction if the next pixel is also a disturbance (note that all pixels on the path are tested, not only those which were part of the initial sample). When it reaches a point which is not a disturbance, it attempts to change direction (i.e. the path 'bounces') to one of the eight horizontal, vertical and diagonal directions, and continues moving. However, it will only try directions which still make progress in the original direction (so, e.g. if it was initially moving to the left, it will only try to move diagonally upwards and leftwards or diagonally downwards and leftwards). This process is then continued until there are no more disturbances in any of the allowable directions. The code also allows flexibility for additional bounces to be carried out for each point (with a starting position randomly chosen from within the current bounding box) to improve the accuracy of the bounding boxes created.

After the bouncing procedure has finished, the maximal and minimal x and y coordinates on the bouncing paths are recorded. Under the assumption that the animal will be an approximately rectangular bounding box, one can then create a bounding box for each path from these maximal and minimal coordinates. If any paths started at separate points have overlapping bounding boxes, these bounding boxes are joined together. The effectiveness of this can be seen in the example of Figure 5—here, the bounding boxes arising from the paths starting at the five points shown in the figure would be combined into a single bounding boxes.

The main benefit of this novel method is its efficiency. By reducing the problem of finding a two-dimensional bounding box to searching for one-dimensional paths, the computational cost of the bouncing procedure scales linearly with the perimeter of the bounding box. This occurs because each path from the starting pixel is always moving towards one edge of the bounding box (either directly, or at a 45-degree angle), and hence, its length is bounded by a constant multiple of the perimeter.

This efficiency means that the code can run quickly even when a substantial proportion of the image is a disturbance. Moreover, as shown in Figure 5, one can recover a good approximation to the bounding box from only a small number of bounces, and hence not all of the disturbance points need to be tested—if they are in an already-formed bounding box by the time it is their 'turn' to be tested, then the bounce procedure is not carried out.



FIGURE 5 An illustration of five realisations of the 'bouncing' procedure. The starting points are shown by red circles, and the bouncing paths are shown by yellow lines. Note that the lines are of width four pixels to improve their visibility, although each considers only a single pixel at a time.

2.2.3 | Bounding box analysis

The final stage of the algorithm analyses each bounding box to determine whether it is likely to be an animal, or simply an inconsequential disturbance.

The first and most important test is the size of the bounding box (i.e. its area), which must be larger than a certain threshold (which can be inputted by the user). Different values of this threshold can be used depending on whether the image was a daytime or a nighttime image, to account for the fact that animals are more similar to the background (and therefore likely to create smaller bounding boxes) at night. However, it is important to note that the size of the animals in the image will vary depending on how far away from the camera the animal is, and that the bounding box size should be chosen sufficiently small that all of these animals will be registered.

The second condition is that the bounding box must contain above a certain threshold of pixels that qualify as disturbances. This can help to remove any large rectangular bounding boxes that are created by small objects which have a diagonal orientation (with respect to the image). Moreover, it can help to remove animals whose colouring is, in the majority, outside the colour and greyscale bounds that have been specified by the user (if, e.g. the user is only interested in specific species).

2.2.4 | Adjacent images

It is often the case that images containing animals occur in sequences (as the animal moves through the field of view of the camera). To account for this, one can choose a value n such that if an image is within n images of an image that has been classed as a positive, and the timestamp of this image differs by at most some threshold (set as default to 20s) from the timestamp of the positive image, then it is also classed as a positive, irrespective of what its classification would have been otherwise. We call these images adjacent images.

Note that adjacent images that would otherwise have been classed as negatives do not trigger their own set of additional positive images. This step is helpful as animals often trigger a long sequence of images, some of which may be misclassified by the

algorithm, and this step helps to find these errors. The default value of n is 1, which strikes a good balance between increasing sensitivity without decreasing specificity to a large degree.

2.2.5 | Extending the set of positive images

An additional consideration is that images containing animals can affect the classification of those images near to them because they may interfere with the background image created, particularly if an animal moves close to the camera. This will be a problem regardless of the difference in times between this image and the images near to it in the image sequence. Thus, we consider 'extending' the set of positive images by including the images either side of each image originally in the set of positives (including adjacent images). In the Results section, we present separately the cases where the data set is and is not extended.

2.2.6 | Algorithm outputs

Sherlock puts images in three categories—those which potentially contain the animal of interest, those which it believes do not contain the animal of interest and images that caused an error when being processed (this is very rare, but may occur if an image has been corrupted). The basic summary of this categorisation is found in CSV (comma separated value) files which it outputs into each folder that contains images. Secondly, Sherlock can create a new copy of each image, with the addition of a red box drawn around anything it believes to be an animal of interest, in a folder based on its categorisation. This may be of use if the potential animal images are then categorised by a human, as it will highlight which disturbances Sherlock has categorised as an animal.

2.3 | Algorithm summary

Sherlock is summarised in Algorithm 1, with some function definitions given in Table 1.

Algorithm 1 Pseudocode for the Sherlock algorithm. Note that this algorithm simply outputs a vector saying whether or not an image contains an animal, and the true code has more detailed outputs. Note that the functions defined in Table 1 are used.

```

Initialise image vector  $I$ 
Initialise background image number  $b = 0$ 
Initialise pixel sample number  $s$ 
Initialise bounce number  $b$ 
Initialise adjacency number  $a$ 
Initialise time threshold  $\tau$ 
Initialise animal indicator vector  $\mathbf{v}$  with each entry as FALSE
for  $i, \mathcal{I}$  in enumerate(I) do
     $\mathcal{I} = I_i$ 
    if  $b == 0$  then
         $\mathcal{B} = \text{BackgroundImage}(\mathcal{I})$ 
        Update  $b$ 
         $\mathcal{B} = \text{number of images used to make } \mathcal{B}$ 
    end if
     $D = []$ 
     $\mathcal{p} = \text{PixelSample}(\mathcal{I}, s)$ 
    for  $p$  in  $\mathcal{p}$  do
        if  $C_p(p)$  then
            Append  $p$  to  $D$ 
             $\triangleright$  pixel  $p$  is a disturbance
        end if
    end for
     $B = []$ 
    for  $p$  in  $D$  do
        if  $p$  in  $b$  for any  $b$  in  $B$  then
             $\triangleright$  If the pixel is in an already processed box
            continue
        end if
         $X = \text{Bounce}(p)$ 
        for bounce = 0, 1, 2, ...,  $b - 2$  do
            Choose random pixel  $q$  in  $X$ 
             $X = \text{Union}(X, \text{Bounce}(q))$ 
             $\triangleright$  Perform initial bouncing step
             $\triangleright$  Additional bounces
             $\triangleright$  Bounce starting point
             $\triangleright$  Add bouncing from  $q$  to  $X$ 
        end for
        Append  $X$  to  $B$ 
    end for
     $B = \text{Overlaps}(B)$ 
     $F = []$ 
    for  $X$  in  $B$  do
        if  $C_b(X)$  then
            Append  $X$  to  $F$ 
             $\triangleright$  Testing bounding box
             $\triangleright$   $X$  classed as an animal
        end if
    end for
    if  $F$  is not empty then
         $v_i = \text{TRUE}$ 
        for  $k = -a, -a+1, \dots, a$  do
            if  $|\text{Time}(I_i) - \text{Time}(I_{i+k})| < \tau$  then
                 $v_{i+k} = \text{TRUE}$ 
                 $\triangleright$  If images were taken close together
                 $\triangleright$  Adjacent also flagged
            end if
        end for
    end if
end for

```

Parameter group	Parameters	Function
Colour constraints	Bounds and greyscale	Boolean $C_p(p)$ on pixels
Bounding box constraints	Size and disturbance %	Boolean $C_b(b)$ on bounding boxes

Classification method	Number (% correctly classified) of		
	Images		Sequences
	With badgers	Without badgers	With badgers
Human	308	240,288	67
Sherlock (unextended)	283 (91.9%)	118,351 (49.3%)	64 (95.5%)
Sherlock (extended)	302 (98.1%)	100,235 (41.7%)	66 (98.5%)

TABLE 3 Sherlock's performance across three pairs of sites, with the majority of images from the first pair being of cattle, the majority in the second pair being of sheep and all of the third pair containing no livestock. Across all sites, 23 of the images contained badgers, of which 20 were correctly identified. Note that these results are based on the unextended set of positives.

Primary animal	Total images	Percentage of images containing livestock	False-positive rate
Cow	3340	68.0%	92.3%
Sheep	14,000	72.6%	66.7%
None	7746	0%	7.7%

2.4 | Code runtime

A timed test of Sherlock 6487 images was carried out on a Dell desktop with Intel Core i7-8665U CPU with four cores and 32GB of RAM. The images were stored on an external hard drive operating at the USB 2.0 interface speed, and each image was a JPEG file between 1 and 2 megabytes in size, containing just over 4 million pixels. The code was run through the Python 3 user interface, Jupyter Lab.

The 6487 images took 6384 s (or approximately 3 h and 15 min) to process, which is a rate of approximately 0.98 s per image.

3 | RESULTS

3.1 | Overall results

Table 2 shows that Sherlock achieved a reasonably good performance, removing close to 50% of the images in the unextended set. Moreover, the fact that only 8.1% of the images and 4.5% of the sequences were missed in the two cases shows that Sherlock's ability to reduce the need for human tagging comes with only a small cost in the reduction of relevant data.

Depending on the exact parameters of the study, it may be beneficial to extend the set of positives. In this case, as one would expect, the percentage of false positives increased from 50.7% to 58.3%. However, almost all (98.1%) of the badger images were correctly

TABLE 1 Functions used in Sherlock to assess disturbances.

TABLE 2 Sherlock's performance on the test data. Both the cases when the set of positive images was extended (i.e. images which were immediately before or after an image classified as a positive were counted as positives) and when the set of positive images was not extended are considered.

identified in this case, highlighting the fact that many of the false negatives from Sherlock were simply a part of a sequence of images of badgers. The fact that only one sequence was missed in this case highlights Sherlock's sensitivity, and suggests that researchers can be confident that the vast majority of the useful data will be retained.

3.2 | Performance by site

Part of the reason for Sherlock's low precision is that many of the sites on which it was tested contained a number of other animals which were not badgers, but which had similar coloration (such as cattle and sheep). **Table 3** shows that Sherlock performed poorly on sites where sheep and cattle are abundant, but performed markedly better on sites where no livestock were present. Indeed, the false-positive rate of 7.7% is very similar to the 7.5% of Zilong (Wei et al., 2020) and shows that Sherlock could be an extremely useful tool on similarly clean data sets.

3.3 | Ability to remove empty images

As shown by the previous section, Sherlock's simplicity means that it is not effective at distinguishing between species of similar colours. Thus, Sherlock may perhaps be most useful at removing 'empty' images from the data set (i.e. those images with no animals or humans in). Note that Sherlock's parameters were fixed at the same values as those used in the larger, badger-specific test, with the exception that no colour or greyscale filters were applied.

As expected, **Table 4** shows that Sherlock achieved a similar, though slightly higher, false-negative rate (11.3%) under this evaluation, while achieving a substantially lower false-positive rate of 7.9%. This false-positive rate is similar to that achieved in the data sets containing no animals considered in **Table 3**, providing further evidence that Sherlock's low precision was simply caused by misidentifying other animals as badgers. Thus, these data suggest that Sherlock could be a valuable preprocessing tool for researchers, substantially reducing the amount of images that need to be classified.

TABLE 4 Sherlock's performance at identifying empty images on a subset of 15091 images which were tagged as 'empty' (which were considered negatives) or 'non-empty' (which were considered positives). Note that these results are based on the unextended set of positives.

Total images	False positives	True positives	False negatives	True negatives
15,091	597	6659	847	6988

TABLE 5 The results of using different parameter values on a test data set containing 1000 images, 68 of which contained badgers. Note that these results are based on the unextended set of positives.

Bounces	Pixel sample size	Run time (minutes)	Precision	Recall
4	5000	31.1	15%	93%
4	1000	16.0	16%	91%
20	5000	67.0	15%	93%
20	1000	34.0	14%	84%

3.4 | Testing with other parameter sets

As shown in Table 5, there was, in general, very little difference between the precision and recall achieved by these algorithms, though the algorithms with smaller sample sizes did have a slightly reduced recall. Most surprising was the drop in precision of the algorithm with a sample size of 1000 and 20 bounces, which performed worse than the algorithm with a sample size of 1000 and 4 bounces. This could have been caused by the increase in randomness in the results which occurs when a smaller sample size is used. It also suggests that there may be little value in increasing the bounce parameter, particularly as it comes with the cost of increased runtime. Indeed, setting this parameter too high could lead to bounding boxes which should be separate being joined together.

The clearest difference between the four parameter sets was in the runtime of the algorithm, with the quickest performing around 75% quicker than the slowest, and with only a small decrease in sensitivity. Further investigation, and a proper definition of optimality would be needed to determine the 'optimal' set of parameters, but it certainly suggests that there may be scope to speed up the algorithm, particularly if one is willing to accept a small drop in sensitivity.

3.5 | Comparison to human tagging

To assess the practical usefulness of Sherlock, two of the authors separately tagged the set of 6531 images with and without the use of Sherlock. The results are summarised in Table 6.

In both cases, Sherlock substantially reduced the human time taken to tag these images. Moreover, Sherlock had only a small effect on the number of images that each person tagged as containing

TABLE 6 The results from two of the authors separately tagging the same set of 6531 images with and without the use of Sherlock.

Tagger	With/without Sherlock	Tagging time (minutes)	Badgers tagged	Sequences
1	Without	20.9	23	6
1	With	13.3	22	6
2	Without	25.8	20	6
2	With	7.8	21	6

a badger and, in one case, this effect was positive as Tagger 2 found an additional image that contained a badger. Indeed, the largest source of error in this process appears to be the human doing the tagging as, given an identical set of images without Sherlock, Tagger 1 tagged 15% more images as badgers than Tagger 2.

3.6 | Comparison to a simple classification algorithm

Finally, we compared Sherlock to a simple tagging algorithm using functions from the Python package OpenCV (Bradski & Kaehler, 2000). To ensure a fair comparison, both algorithms used the same method of forming background images. The simple algorithm compared each image to the background image using the *cv2.absdiff* function, before converting this image, first to a greyscale image (using *cv2.cvtColor*), and then to a binary threshold image (using *cv2.threshold*). Tests were carried out with different values of the threshold. Finally, the *cv2.findContours* function was used to locate contours in this threshold image, and the size of the largest contour (found using *cv2.contourArea*) was recorded.

For this algorithm to classify images, it is necessary to choose a 'size value' for how large a contour needs to be in order for the image to be classed as containing an animal. In the below comparisons, the best possible size value was used for each of the OpenCV algorithms (i.e. the value which maximised the number of images that were correctly classified). However, Sherlock's parameters were fixed at the same values as those used in the larger, badger-specific test, with the exception that no colour or greyscale filters were applied. Table 7 shows Sherlock outperformed the basic algorithm for all tested values of the threshold value and all size values. It is also notable that it had by far the lowest number of false negatives (of course, choosing a different size value for the OpenCV algorithms would change this, but would make them less accurate overall). Thus, it appears that there is a substantial advantage in using Sherlock compared to a more basic algorithm.

Of course, the added complexity of Sherlock comes with an additional computational cost. As previously discussed, Sherlock took 0.98 s per image while the OpenCV algorithms ran in just over half the time, taking 0.59 s per image. However, part of the reason for this is that the in-built OpenCV functions have been highly optimised for speed, and we believe there is scope for Sherlock to be rewritten to run in a similar timeframe to the OpenCV algorithm.

Algorithm	Total images	FP	TP	FN	TN	% correctly classified
Sherlock	15,091	597	6659	847	6988	90.5%
OpenCV(60)	15,091	362	6191	1315	7222	88.9%
OpenCV(75)	15,091	291	6174	1332	7294	89.3%
OpenCV(90)	15,091	404	6148	1358	7181	88.3%

TABLE 7 The results from comparing a simple algorithm, using OpenCV functions, to Sherlock. The same data set as Table 4 was used, with 'positive' images being defined as those containing animals or humans. The number after each OpenCV algorithm denotes the threshold used in the *cv2.threshold* function.

4 | DISCUSSION

Sherlock has the potential to substantially reduce the number of human hours required to process a camera-trapping data set, as it can remove a large proportion of unwanted images while losing very few images of interest. It has performed well across a range of camera placements which contained many complicating factors such as vegetation and other animals that were not of interest. Moreover, an almost identical algorithm has been used by one of the authors in their work for a semi-professional football club for tracking players from video footage of their matches, highlighting the algorithm's flexibility and the potential for its widespread adoption. It is simple to set up, with detailed instructions available on our GitHub page (Matthew, 2023), and does not require high levels of computing power to process images at a reasonably quick rate.

Because of its low computational cost, an area for future research would be to investigate whether a version of Sherlock could be deployed directly on camera traps, meaning that empty images could automatically be discarded. This would reduce the risk of the camera's memory becoming full over the course of its deployment, particularly if it were placed in an area with high levels of vegetation, where the vast majority of images do not contain animals.

Across the whole data set, Sherlock did not achieve as high a specificity as some similar algorithms. For example, Zilong (Wei et al., 2020) removed approximately 92.5% of the empty images from its data set, compared to the 49.3% that Sherlock achieved from the unextended set of positives. However, such a direct comparison may not be an appropriate assessment of the relative utility of the two algorithms. When sites with no livestock were considered, Sherlock was able to remove 92.3% of the images, which is very similar to Zilong's performance. This suggests that Sherlock's apparent low performance was simply due to the high levels of noise in the data, and that it has the potential to be an extremely useful image-removal tool, particularly when the camera trap has little interference from other animals, or when all animals are of interest.

However, most importantly, the fact that Sherlock has such a high sensitivity of 91.9%—higher than, for example, the reported 87% of Zilong—means that it could be used as a tool to preprocess a large set of data, reducing the number of images that need to be classified by more computationally intensive algorithms such as MegaDetector which, for example, could be more accurately used to classify animals to species level. The high sensitivity means that very little useful information would be lost during this step, while

the computational savings may be valuable, both in terms of financial and time cost. This would be particularly relevant in the examples considered in Table 3, where, in the sites with sheep, Sherlock was able to remove a reasonable proportion of images from a very noisy data set.

There are a number of ways in which Sherlock could be refined to increase its performance. Primarily, adding the option to parallelise the algorithm, and in particular adding the option to run it on a GPU, would lead to a large reduction in the amount of time taken to process each image (at the expense of making it less accessible). Moreover, it may be possible to more intelligently sample pixels from the image, so that the perspective of the image is taken into account—one needs to sample fewer pixels in the foreground than the background to achieve the same level of coverage. Combining this with position-dependent-bounding box size restrictions could further improve performance, particularly if there is vegetation close to the camera that frequently triggers it.

There are also some limitations to the method of forming the background images. Firstly, there is not always an immediate transition between daytime and nighttime images, and so there may be some images taken at dawn or dusk which are very dissimilar to the background. These images will then be classified as potentially having an animal, adding to the number of false positives in the data set. However, this was true for a small proportion of the images and so did not substantially affect algorithm performance.

Moreover, using the median image only works when the background is truly fixed, otherwise there may be a number of pixels which are misclassified as disturbances. For example, if a blade of grass constantly sways in the wind, it should be possible to use a background comparison method which can take into account the possibility that the background may oscillate between multiple states. However, such improvements would come with a potentially substantial extra computational cost.

A final limitation is that the bouncing procedure was designed with convex animals, such as badgers, in mind. When tested on football players, the algorithm struggled to detect arms and legs (although could still find the players from their torsos), with a large number of bounces required to do this successfully. Thus, Sherlock may struggle to give accurate bounding boxes of animals such as giraffes, with many long thin sections, although it should still be able to detect their presence. We hope to investigate this further in future work.

Despite these limitations, we believe that, in its current form, Sherlock has the potential to be of use to a wide range of researchers.

The levels of specificity and sensitivity that it achieves can substantially reduce the amount of processing by humans required in camera-trapping projects. Moreover, its ease of use and minimal hardware requirements mean that it is available to the vast majority of researchers, and we hope that its adoption can help accelerate many areas of biological research.

AUTHOR CONTRIBUTIONS

Matthew J. Penn and Verity Miles were involved in conceptualisation and writing—original draft. Verity Miles was involved in data curation and validation. Verity Miles, Kelly L. Astley and Cally Ham were involved in investigation. Matthew J. Penn was involved in methodology and software. Matthew J. Penn, Verity Miles, Rosie Woodroffe, Marcus Rowcliffe and Christl A. Donnelly were involved in writing—review and editing.

FUNDING INFORMATION

Matthew J. Penn's work on this paper was funded by a EPSRC DTP studentship, awarded by the University of Oxford to fund his DPhil in Statistics. Verity Miles' work on this paper was funded by the NERC Science and Solutions for a Changing Planet Doctoral Training Programme, Imperial College London, grant number NE/S007415/1. Cally Ham's work was funded by the Cornwall Wildlife Trust and the NERC Science and Solutions for a Changing Planet Doctoral Training Programme, Imperial College London, grant number NE/L002515/1. The funders had no role in the design of this study, its execution, analyses, interpretation of the data or the decision to submit results.

CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

PEER REVIEW

The peer review history for this article is available at <https://www.webofscience.com/api/gateway/wos/peer-review/10.1111/2041-210X.14254>.

DATA AVAILABILITY STATEMENT

The code for Sherlock is available on Zenodo at <https://zenodo.org/records/10021195> (Miles, 2023). The underlying data were too large to upload to a single Zenodo repository, but can be found by following the links listed in this Zenodo repository.

ORCID

Matthew J. Penn  <https://orcid.org/0000-0001-8682-5393>

Cally Ham  <https://orcid.org/0000-0002-9281-038X>

Rosie Woodroffe  <https://orcid.org/0000-0003-2104-3133>

REFERENCES

Adam, M., Tomášek, P., Lehejček, J., Trojan, J., & Jůnek, T. (2021). The role of citizen science and deep learning in camera trapping. *Sustainability*, 13(18), 10287.

- Beery, S., Morris, D., & Yang, S. (2019). Efficient pipeline for camera trap image review. *arXiv Preprint arXiv:1907.06772*.
- Bradski, G., & Kaehler, A. (2000). OpenCV. *Dr. Dobb's Journal of Software Tools*, 3, 2.
- Burton, A. C., Neilson, E., Moreira, D., Ladle, A., Steenweg, R., Fisher, J. T., Bayne, E., & Boutin, S. (2015). Wildlife camera trapping: A review and recommendations for linking surveys to ecological processes. *Journal of Applied Ecology*, 52(3), 675–685.
- De Bondi, N., White, J. G., Stevens, M., & Cooke, R. (2010). A comparison of the effectiveness of camera trapping and live trapping for sampling terrestrial small-mammal communities. *Wildlife Research*, 37(6), 456–465.
- Green, S. E., Rees, J. P., Stephens, P. A., Hill, R. A., & Giordano, A. J. (2020). Innovations in camera trapping technology and approaches: The integration of citizen science and artificial intelligence. *Animals*, 10(1), 132.
- Kucera, T. E., & Barrett, R. H. (2011). A history of camera trapping. In *Camera traps in animal ecology* (pp. 9–26). Springer.
- Matthew, J. P. (2023). *Sherlock*. <https://github.com/mpenn114/Sherlock>
- Meek, P. D., Ballard, G., Falzon, G., Williamson, J., Milne, H., Farrell, R., Stover, J., Mather-Zardain, A. T., Bishop, J. C., Cheung, E. K.-W., Lawson, C. K., Munezero, A. M., Schneider, D., Johnston, B. E., Kiani, E., Shahinfar, S., Sadgrove, E. J., & Fleming, P. J. S. (2020). Camera trapping technology and related advances: Into the new millennium. *Australian Zoologist*, 40(3), 392–403.
- Microsoft. (2022). *Cameratraps/megadetector.md at main · Microsoft/Cameratraps*. <https://github.com/microsoft/CameraTraps/blob/main/megadetector.md>
- Miles, V. (2023). *Sherlock Camera Trap Dataset*. <https://doi.org/10.1101/2023.03.08.531714>; <https://zenodo.org/records/10021195>
- 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.
- Rowcliffe, J. M., Carbone, C., Kays, R., Kranstuber, B., Jansen, P. A., Meek, P., & Fleming, P. (2014). Density estimation using camera trap surveys: The random encounter model. In *Camera trapping: Wildlife management and research* (pp. 317–324). CSIRO Publishing.
- Rowcliffe, J. M., Field, J., Turvey, S. T., & Carbone, C. (2008). Estimating animal density using camera traps without the need for individual recognition. *Journal of Applied Ecology*, 45, 1228–1236.
- Santosh, D. H. H., Venkatesh, P., Poornesh, P., Rao, L. N., & Kumar, N. A. (2013). Tracking multiple moving objects using gaussian mixture model. *International Journal of Soft Computing and Engineering (IJSC)*, 3(2), 114–119.
- Schneider, S., Greenberg, S., Taylor, G. W., & Kremer, S. C. (2020). Three critical factors affecting automated image species recognition performance for camera traps. *Ecology and Evolution*, 10(7), 3503–3517.
- Schneider, S., Taylor, G. W., & Kremer, S. (2018). Deep learning object detection methods for ecological camera trap data. In *2018 15th conference on computer and robot vision (CRV)* (pp. 321–328). IEEE.
- Tabak, M. A., Norouzzadeh, M. S., Wolfson, D. W., Newton, E. J., Boughton, R. K., Ivan, J. S., Odell, E. A., Newkirk, E. S., Conrey, R. Y., Stenglein, J., & Iannarilli, F. (2020). 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.
- van Lunteren, P. (2023). EcoAssist: A no-code platform to train and deploy custom YOLOv5 object detection models. *Journal of Open Source Software*, 8(88), 5581. <https://doi.org/10.21105/joss.05581>
- Villa, A. G., Salazar, A., & Vargas, F. (2017). Towards automatic wild animal monitoring: Identification of animal species in camera-trap

images using very deep convolutional neural networks. *Ecological Informatics*, 41, 24–32.

Wei, W., Luo, G., Ran, J., & Li, J. (2020). Zilong: A tool to identify empty images in camera-trap data. *Ecological Informatics*, 55, 101021.

Xi, T., Wang, J., Qiao, H., Lin, C., & Ji, L. (2021). Image filtering and labeling assistant (IFLA): Expediting the analysis of data obtained from camera traps. *Ecological Informatics*, 64, 101355.

XnView MP. (2020). Windows version 1.0. <http://www.xnview.com>

How to cite this article: Penn, M. J., Miles, V., Astley, K. L., Ham, C., Woodroffe, R., Rowcliffe, M., & Donnelly, C. A. (2024). Sherlock—A flexible, low-resource tool for processing camera-trapping images. *Methods in Ecology and Evolution*, 15, 91–102. <https://doi.org/10.1111/2041-210X.14254>