Bird Counting

1. The data

There are images from bird surveillance. We want to count some specific species (shags and cormorants). There are other species in the images. The images look lie that in Figure 1.

Figure 1.

We have annotations in the form of bounding boxes for each image, as in Figure 2.

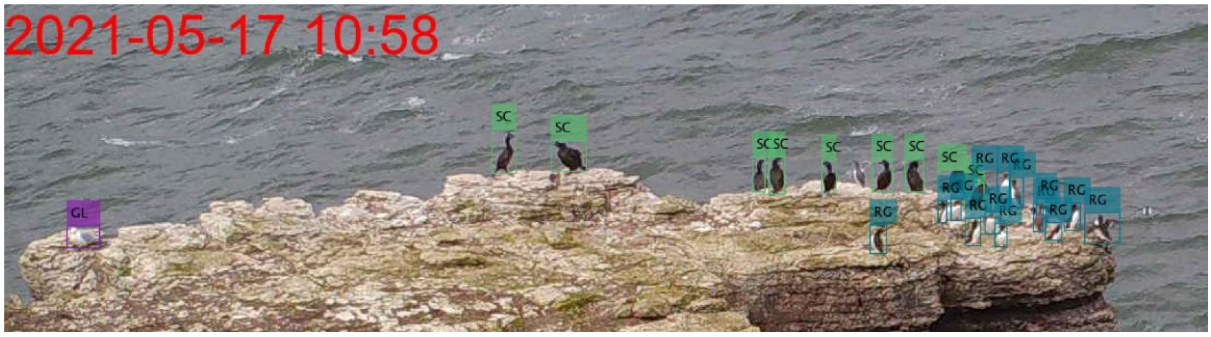

Figure 2

For the moment, we want to count ALL the birds. If successful, we will try to discriminate between the species and count only the species of interest.

Figure 3 shows example of the bounding boxes with their content. The examples were taken randomly across the images. The resolution is not great, unfortunately. There are a total of 245 images with 2993 bounding boxes. Some images don't have any birds.

2. Deep learning models

We want to do object detection (bounding box detection) so that we can count the birds. The exact match of bounding box to bird is not that essential, as long as we have the right number of boxes in the image. The class labels are ignored, and all bounding boxes are taken to be "Bird".

Deep learning modes are currently the best object detectors, if trained properly!

https://paperswithcode.com/task/object-detection

Figure 3

The top detectors on the list are not easily available as code, even though the website assumes they are. Frank Krzyzowski has been trying to install them starting from the best model – with little success thus far. On the other hand, there are some older (and still good) models such as

- ACF Aggregated Channel Features detector
- RCNN Regions with convolutional neural networks
- Fast RCNN
- Faster RCNN
- YOLO You only look once

There are examples on the MATLAB website. All examples are about pedestrian, vehicle or road sign detections. There are also pre-trained models for these type of tasks.

I have tried some of the models. The trained versions need fine-tuning with our data, which sometimes takes over two hours. Unfortunately, the results were not very encouraging. An example of the result from the ACF detector is shown in Figure 4.

Figure 4

The models were tried with the resubstitution protocol, which means that the tuning was done on the whole data, and the results are checked on the same data. At the moment, I don't care about overfitting. If the detector CAN overfit, I will be happy. If the detector cannot recognise the bounding boxes it was trained upon (refined, rather), then the detector is not of use.

The pretrained YOLO does not recognise anything.

For RCNN, I used the MATLAB example on their website, took the bare bones, and made my own version with the bird images. The RCNN detector steps on a pretrained CNN, which I took from the example on the website. It had been trained on the CIFAR10 data (a detail). The RCNN detector took about 2 hours to train on the bird data set. The results are not particularly great. Figure 5. shows an example.

Figure 5.

The Average Precision is 20%, which means roughly that 20% of all detections match true bounding boxes. More to the point, the count of bounding boxes does not match between the original annotation and the detections. Figure 5 shows this.

Fast RCNN and Faster RCNN are not meant to improve accuracy; only training speed. (I did try to adapt the MATLAB example with the Faster RCNN but it is too idiosyncratic about vehicle detection.)

Data augmentation could help with the training. and Some post-processing of the bounding boxes could help in sifting out the smallest and the largest boxes, as well the ones that are not in a feasible position. But this will hardly improve the success by much.

3. Handcrafted feature models

Deep learning aside, I tried using "old-fashioned" crafted features by implementing the following steps:

1. Store all clips using size 30x40 pixels (width-x-height). This is a reasonable guess of a bounding box, judging by the distribution of height and width.

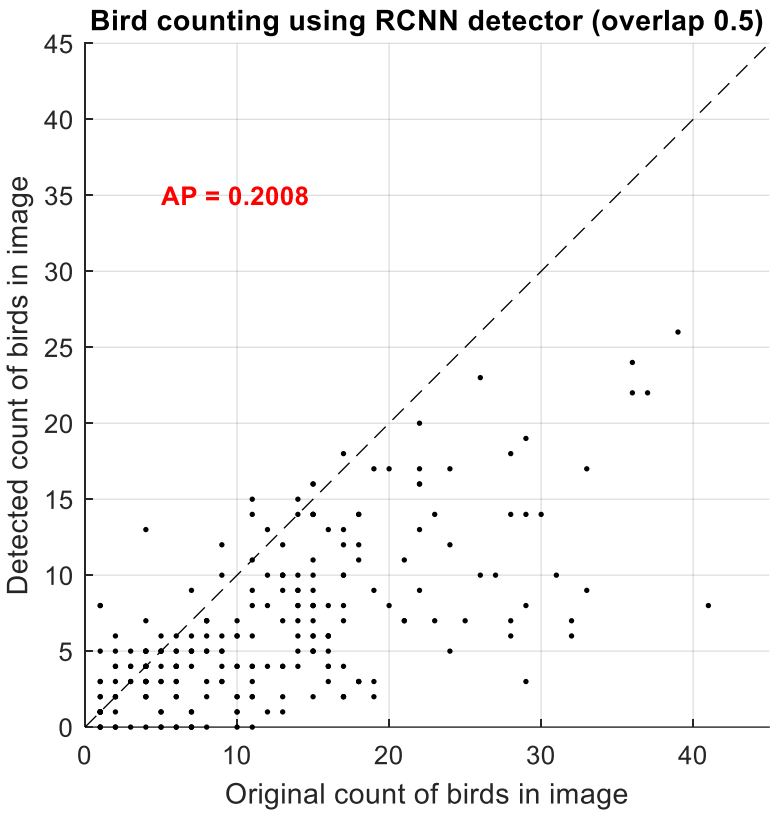

Figure 5.

2. Read each clip, convert to grey, flatten it, and add it as a row in the dataset.

3. Create a set of negative examples by choosing random images and cropping bounding boxes of the same size as the clips at random feasible locations. Add the examples to the data.

3. Run PCA and keep 30 components as the features. HOG features were also extracted but were later found to be inferior to the PCA features.

4. Train and test classifier models on the data created. (The results are shown in Table 1.)

5. Apply a chosen classifier on the images (LDC) by checking every possible bounding box location and classify the crop as BB or background. Tally the BB classifications by creating an array of zeros of the image size, and subsequently adding a number to all pixels that belong to the crop. We add 1, if the box is classified as BB and zero otherwise. Alternatively, we add the probability (certainty) that the crop is a BB. Thus, we have a heat image of bounding boxes. An example is shown in Figure 6.

6. Choose a threshold for the heat image, find the number of connected components and check the results against the true number of boxes.

Table 1. Classification accuracies using 10-fold cross-validation

--Classifier--	PCA30	HOG
Linear Discriminant Analysis	0.9445	0.8777
3-nn	0.9657	0.7897
Decision Tree	0.9360	0.7857
Bagging	0.9559	0.8378
Random Forest	0.9621	0.8662

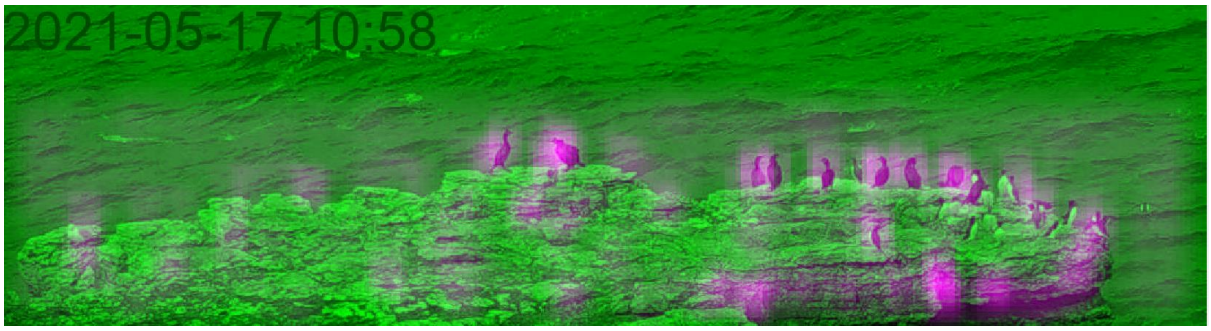

Figure 6

Figure 7 shows the heatmap converted into detections for two different thresholds.

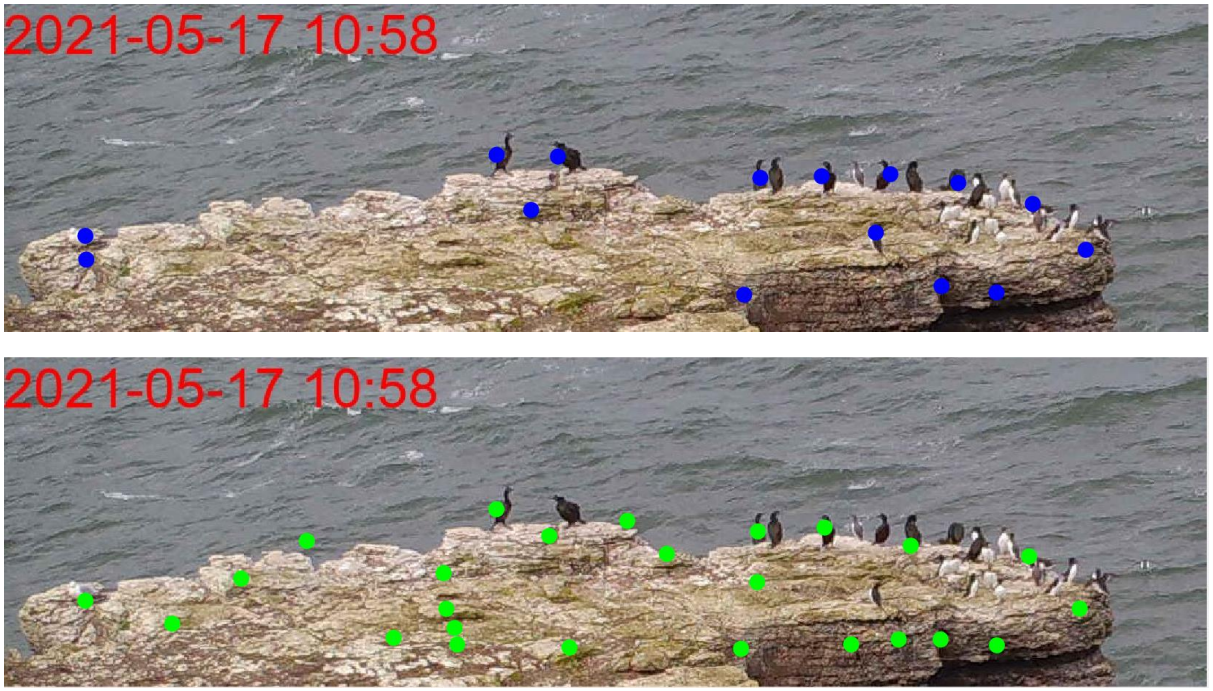

Figure 7

Figure 8 shows the correspondence of detected and true bird counts.

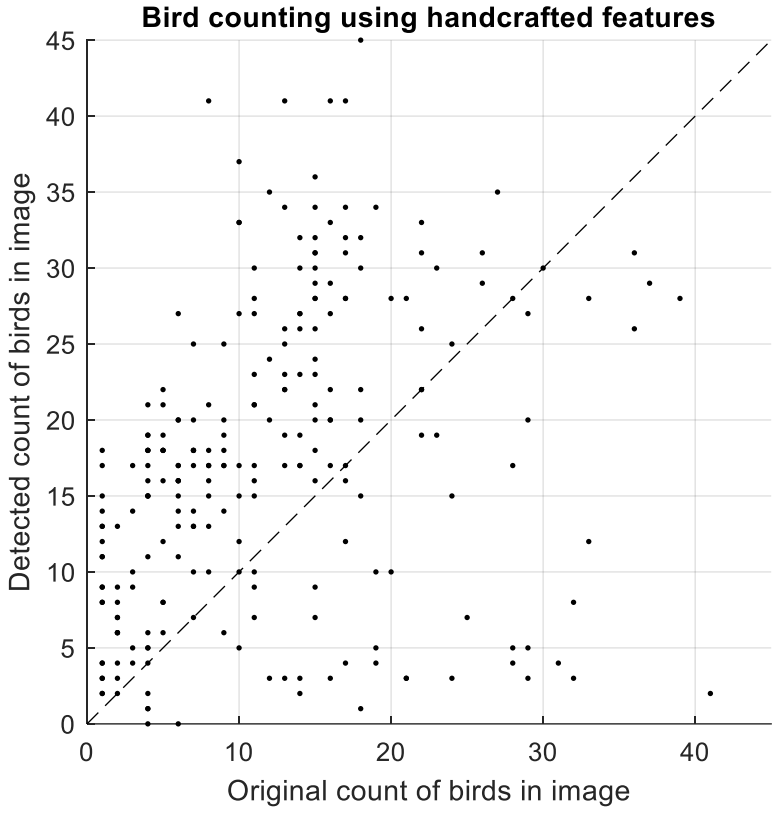

Figure 8

4. Conclusion

As an overall conclusion, I could not find a reasonable Deep learning model or handcrafted feature model to recognise or count the bounding boxes correctly. The accuracy is too low to warrant further effort.