TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Liis Harjo 153005IAPM

DETECTING AND COUNTING PENGUINS IN IMAGES WITH THE YOLO APPROACH

Master's thesis

Supervisor: Ago Luberg
MSc

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Liis Harjo 153005IAPM

PINGVIINIDE TUVASTAMINE JA LOENDAMINE PILTIDEL KASUTADES YOLO LÄHENEMIST

Magistritöö

Juhendaja: Ago Luberg
MSc

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Liis Harjo

07.05.2018

Abstract

This thesis is about detecting and counting penguins in images taken with fixed cameras at different locations in Antarctica. There exist different approaches for counting objects in images. This thesis proposes a detection-based approach to count the penguins in the penguin dataset. This thesis describes how to use the penguin dataset with the YOLO algorithm to train different machine learning models. The results of the different models are evaluated and presented.

This thesis is written in English and is 51 pages long, including 6 chapters, 9 figures and 5 tables.

Annotatsioon

Pingviinide tuvastamine ja loendamine piltidel kasutades YOLO lähenemist

Antud magistritöö käsitleb pingviinide tuvastamist ja loendamist piltidel, mis on tehtud kinnitatud kaameratega erinevates asukohtades Antarktikas. Objektide loendamiseks piltidel eksisteerib erinevaid lähenemisi. Antud magistritöö kasutab pingviinide loendamiseks tuvastus-põhist meetodit. Antud magistritöös kirjeldatakse, kuidas kasutada pingviinide andmestikku ja YOLO algoritmi erinevate masinõppe mudelite treenimiseks. Esitatakse erinevate mudelite tulemused koos hinnangutega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 51 leheküljel, 6 peatükki, 9 joonist, 5 tabelit.

List of abbreviations and terms

CNN	Convolutional neural network
RELU	Rectified linear unit

Table of contents

1 Introduction 10
2 Related Work 13
2.1 Detection-based methods for counting objects in images 13
2.2 Density-based methods for counting objects in images 14
2.3 Methods that combine detection and density estimation 16
2.4 Other approaches for counting objects in images 17
2.5 Counting in the Wild 20
2.6 YOLO algorithm 23
3 Methodology 25
3.1 Dataset 25
3.2 Generating bounding boxes 27
3.3 Training the model 31
3.4 Evaluating the results 32
4 Results 34
4.1 Results of different models 34
4.2 Results of model 2 39
4.3 Results of model 6 42
4.4 Results of model 7 43
5 Discussion 45
6 Summary 49
References 50
Appendix 1 – Source code of the models 51

List of figures

Figure 1 Example of an image where smaller generated bounding boxes work well 29
Figure 2 Example of an image where smaller generated bounding boxes do not work well.... 30
Figure 3 Example of larger generated bounding boxes 31
Figure 4 Example of an image where the evaluation algorithm does not work correctly 33
Figure 5 Example of model 6 performance 37
Figure 6 Example of model 7 performance 38
Figure 7 Example of model 3 performance 39
Figure 8 Example image from location HALFc 41
Figure 9 Example image from location PETEf.... 42

List of tables

Table 1 Statistics about the different locations of the penguin dataset 27
Table 2 Results of different models 36
Table 3 Results of model 2 40
Table 4 Results of model 6 42
Table 5 Results of model 7 43

1 Introduction

In many applications there is a need to count objects in images. Some of the examples include monitoring crowds, counting cells in microscopic images, and performing wildlife census. Since counting is labour-intensive and humans tend to make mistakes there is a need to automate this process.

One way to automate this process is to use machine learning. Models are trained on images where object counts or locations have been provided. Given a new image, the model will hopefully output accurate object counts or locations for that. Training good models to count objects presents many difficulties. Training computer vision models requires a lot of annotated training data. Nowadays gathering a lot of data may be easy, but a human still needs to annotate each of the training images, which is labour-intensive.

There is the question of what should the model output. Are we interested in plain object counts or is the location of the objects also important, meaning that the model should detect each instance? Obtaining plain object counts may be an easier task, but locations of individual instances may provide additional information – for example about the relationships between objects.

Depending on what is expected as model output, different approaches have been used. Density-based methods predict object density based on an input image and have been shown to provide relatively accurate object counts. But they do not provide information about object locations. In detection-based methods an object detector is used on different parts of an image. Detection-based methods have the benefit of providing object locations, but there is a limitation – usually, these do not work well when objects are overlapping or crowded together.

One area where the need to count objects in images arises is research of Antarctic penguins. Zoologists are interested in sizes of different penguin colonies and how they change over time. In order to monitor different penguin colonies, they are periodically

taking pictures with fixed cameras. This results in a huge number of pictures. There is a need to count and possibly locate individual penguins on these images.

The annotating of these images has been conducted on a public website [1] and is still ongoing. There, any interested person can join and help with locating individual penguins on the images. This annotation process has resulted in a dataset of annotated images that can be used for training and testing machine learning models. Part of the dataset has been made publicly available at [2].

The goal of this thesis is to train a machine learning model in order to automate the process of counting penguins in images. Since convolutional neural networks have performed well for many computer vision applications, the approach used in this thesis is based on them. The aforementioned dataset is used for training the model and evaluating its performance. In "Counting in the wild" [3] the same dataset was used for the same purpose. In this paper, the researchers have used a density-based method. The approach of this thesis is detection-based –the YOLO [4] algorithm is used to detect penguins on images.

There are several difficulties. The dataset is quite challenging – the scale and perspective of the images varies a lot. On images, the birds are often crowded and there are challenging weather conditions. Since the images have not been annotated by experts, but by ordinary people, the annotations contain a lot of mistakes. Especially when there is crowding, the annotators have missed a lot of the penguin instances.

In the dataset penguin instances have been marked by dot annotations. YOLO algorithm requires bounding boxes as input. In order to use the penguin dataset with the YOLO algorithm, bounding boxes are generated from dot annotations. This causes additional difficulty when evaluating the results. Since the model outputs bounding boxes, but the ground truth is provided as dot annotations, there is a question of which detections should be considered successful.

Chapter 2 presents an overview of existing approaches that deal with counting objects in images. Density-based and detection-based methods are described. Also, additional ways to improve counting models are described – fine-tuning models to new scenes and counting objects interactively. An overview of the paper that deals with training a model with the penguin dataset and an overview of the YOLO algorithm are presented.

Chapter 3 includes a description of the dataset and how it was processed before training the models. It presents the algorithm that was used for generating the bounding boxes, the different parameters that were used when training the models, and the algorithm that was used when evaluating the performance of a model.

Chapter 4 presents the results of the experiments. The results of different models are presented. The penguin dataset contains images from different locations which have different properties. Therefore, the results for different locations are also presented. This is followed by a chapter with a discussion of the results.

2 Related Work

This thesis deals with counting objects in images. It is useful for many applications – cell counting in microscopic images, monitoring crowds, performing wildlife census, counting the number of trees in an area [5]. Therefore, a lot of work has been done in this research area. Supervised methods that deal with counting objects in images can be divided into two categories: counting by detection and counting by density estimation [5]. Counting by detection means that an object detector is used on different parts of the image, where it tries to detect the object. Counting by density estimation means that a density function between image characteristics and object density is learned. Integrating this density function over some image region gives object count for that region. Counting by density estimation avoids detecting individual object instances.

Which approach is more appropriate depends on the degree of overlap between object instances [6]. Detection-based methods may perform well on images where objects have low density and do not overlap frequently. On such images, density-based methods can hallucinate small object counts on image regions that do not contain any objects. Detection-based methods have the additional benefit of locating individual instances. On the other hand, detection-based methods may fail with high-density images where there is a lot of overlap and inter-occlusion between objects. In such cases density-based methods may perform better. That is because such methods use texture matching between the test image and the training set, which can be accomplished even if individual instances are not distinguishable. Many real-life applications may need to handle both scenarios and both scenarios can co-exist on the same image.

2.1 Detection-based methods for counting objects in images

One detection-based approach is described in [7]. The authors explain that detection methods learn object classifiers from labelled training set and then the classifier is applied to different sub-windows and locations of the test image. When there are occlusions between objects, part based representations can be learned. On test image, part detection responses are combined.

The authors note that for detecting objects with partial occlusions, it is possible to use part based detectors. Part detectors are applied to overlapping windows. Since the windows are classified separately, one local feature can contribute to multiple detections. This means that there is a need to merge part detection responses to get object hypothesis. Using part detectors involves some difficulties. Since local features may not be discriminative enough, false detections may occur. Because of occlusion, some object parts may not be detected. At test time, part detectors are applied to an input image and their responses are merged by using some clustering method.

In their approach, a part hierarchy is defined for an object class. Each part is a sub-region of its parent. For each part a detector is learned. A child node inherits image features from its parent node. If target performance cannot be achieved, more features are added to the child node. Also, a pixel-level segmentor is learned for the whole object. At test time part detectors are applied to a new image and image pixels that contribute to detection responses are extracted.

Based on part detection responses, object hypothesis are proposed. Whole-object segmentor is applied to each hypothesis and the silhouette is extracted. Joint analysis is used to enforce the exclusiveness of low-level features, this way one image feature can contribute to at most one hypothesis.

2.2 Density-based methods for counting objects in images

Density-based methods learn a mapping between image features and object counts.

One density-based approach is described in [8]. In this paper, the authors deal with estimating the number of people in an image. They have chosen not to use a detection based method, because, as they state, these do not work well when there are viewpoint or illumination changes. They develop a system to count pedestrians in crowds. The authors point out that existing methods that estimate crowd size use models that do not have spatial information. Since people who are farther from the camera appear smaller, they contribute less in these models. Therefore, they develop a method that uses feature normalization to deal with perspective projection and camera orientation.

As features for their model they use foreground regions given by a background subtraction algorithm and an edge orientation map, which is generated by an edge

detector. For each frame a foreground mask is generated and an edge detector is applied. Total edges length can be used to represent crowd density.

The authors make two assumptions – that all the pedestrians have similar size and that they all lie on a horizontal ground plane. Pedestrians are modelled by a simple cylinder model. The projected 2D height varies when people move on the ground plane and it is possible to measure the ratio of the projected height. The ratio reflects the relative crowd density on the ground plane. Homography between ground plane and image plane is used to estimate the density. After estimating the homography, relative density is computed for each pixel.

The authors train the model on a feed-forward neural network to find the relationship between image features and the number of pedestrians in an image. Estimating the pedestrian count is more difficult when there is significant occlusion. Then the relationship between the features and the number of pedestrians is not linear. To capture this nonlinearity the authors use a neural network with a single hidden layer. The single output unit of the neural network is the estimated crowd count. The authors show with their experiments that a neural network has better estimation results than a linear model.

Another approach that is based on learning a mapping between image features and object counts is described in [5]. Their training images are annotated by dot annotations – object positions are specified by single dots on each object instance. The authors point out that dotting is a natural way to count objects and therefore, for humans, providing dot-annotations is not harder than providing raw counts. In addition, spatial arrangement of the dots can provide additional information.

The authors develop a framework for counting objects in images. In their approach, a density function is generated for an input image. This density function is a real function of pixels. Integrating the density function over an image region gives an estimate of the count of objects in that region.

The authors point out that there is a conceptual difficulty when considering the density function – it is easy to reason about average densities over image regions, but "the notion of density is not well defined at pixel level" [5]. Therefore, given a set of dot-annotations, it is ambiguous what should be the ground truth density.

Another density-based approach is described in [9]. This paper is concerned with counting cells in microscopic images, but the developed methodology could be used in other counting applications as well. Their approach is based on density estimation and they cast the cell counting problem as learning a mapping between an image and a density map. Integrating the density map over an image region gives the estimate of the number of cells in that region. The authors have chosen to use a convolutional neural network.

In their dataset, the ground truth is provided as dot annotations. In training, each dot is represented by a Gaussian and a density surface is formed by the superposition of these Gaussians. The task is to regress the density surface from an image. To achieve this, a CNN is trained. Mean squared error between the output heat map and the target density surface is used as the loss function. Then, for an unseen image, the CNN predicts the density heat map.

In their network, they use convolution-RELU-pooling layers. Then, to undo the spatial reduction, they use up-sampling-RELU-convolution. In their implementation they use back-propagation and stochastic gradient descent for optimization. To increase the amount of data for training, they cut large images into patches and each patch is also normalized. The authors note that the Gaussian-annotated ground truth must be scaled, for example by multiplying it by 100. This is because most of the pixels in the ground truth are in the background and are labelled zero, which causes the network to focus more on the background zero. After pre-training with patches, they fine-tune the parameters with whole images.

2.3 Methods that combine detection and density estimation

There also exist approaches that combine detection and density estimation. One of these is described in [6]. Their method is able to detect groups of objects of different integer sizes. The method avoids discerning individual objects when they are clumped together and at the same time is able to enforce the fact that each group has an integer number of objects.

Another approach that is a mixture of detection and density estimation is described in [10]. In this paper, the authors note that object detection and counting are two related

but also different research topics. Detection approaches tend to be less accurate when objects are overlapping. In such scenarios counting methods tend to do better, but their output is only the number of objects.

For many applications, accurate total counts and locations of objects are equally important. The authors state that in the case of migrating birds, total count shows a global trend for the whole scene and locations of each instance indicate the grouping structure and relationship among them. “Solving the counting and detection tasks together in a unified framework can provide more accurate output in terms of both counting and detection.” [10]

In the authors opinion, previous detection approaches are not suitable for counting-detection task, especially for partially occluded small instances. Small instances grouped together present several difficulties. Many discriminative features and details are blurred or hidden. Perspective makes that problem worse. For detection-based methods, the low resolution of training examples presents a difficulty. There can be heavy overlapping between objects moving in a group and different poses and view-points make the objects appearance look very different.

Density-based approaches avoid individual-level detection and achieve good results in terms of counting. The authors use [5] as inspiration and propose a novel detection framework using object density maps. This framework can output both counting and detection results. The authors make four contributions. They develop a 2D programming method that recovers 2D object locations from object density map. They add global count constraint to the integer programming objective function. They propose a method to estimate the bounding box of the object given a detected object location. The proposed detection method achieves state-of-the-art results on different datasets.

2.4 Other approaches for counting objects in images

When a model is learned on one dataset, it may not perform well on an unseen scene. This problem is addressed in [11]. The authors propose a deep convolutional neural network which is trained with two learning objectives – crowd density and crowd count.

To handle unseen scenes, they present a method to fine-tune the trained CNN to the target scene.

The authors point out that crowd counting is challenging because of occlusions, scene perspective distortions and diverse crowd distributions. Most existing methods are scene specific. For a new scene, the models need to be retrained with new annotations. There aren't many works focusing on cross-scene crowd counting.

The authors propose a framework for cross-scene crowd counting where no new annotations are needed for a new target scene. They propose a CNN based framework that learns a mapping from images to crowd counts. The CNN is trained with a fixed dataset. Then a data-driven method is used to fine-tune the learned CNN to an unseen target scene. Their model is trained with two learning objectives – crowd density maps and crowd counts. Two objectives can assist each other to obtain a better local optimum. In their framework, there is no need for extra labels for a new target scene. The pre-trained CNN is fine-tuned to a new target scene.

The authors use data-driven approach for scene labelling. Labels are transferred from training images to test images by retrieving the most similar training images and matching them with the test image. For an unseen target scene, the authors retrieve similar scenes and crowd patches from the training scenes.

Their CNN model learns a mapping between X and D, where X is a set of low-level features extracted from training images and D is the crowd density map for the image. Density map is created from pedestrians' location, body shape and perspective distortion. These density maps are treated as ground truth for the CNN model. The total crowd number is calculated by integrating over the density map.

Input for their CNN model is image patches cropped from training images. Their model contains three convolutional layers and three fully connected layers. After first and second convolutional layer, max pooling layers are used. Rectified linear unit is used as the activation function.

Their model optimizes alternatively the density map estimation task and the count estimation task. Because of pooling layers the output density map is down-sampled and therefore, the ground truth density map must be down-sampled as well. The count is

calculated by integrating the density map. Two tasks assist each other and enable a better result. Density map is optimized first, because it can introduce more spatial information to the model. After first objective converges, the model optimizes the second task. Two objective losses should be normalized to similar scales. Switch learning approach obtains better results than multi-task learning approach.

The CNN is pre-trained on all training scene data. Each query scene has unique scene properties which affect the performance of the model. Therefore, fine-tuning scheme is adopted to adapt the CNN model to unseen target scenes. Given new target scenes, samples with similar properties are retrieved from the training scenes and added to training data to fine-tune the CNN model.

Another way to deal with unseen target scenes is an interactive system. This approach is described in [12], where the authors present an interactive counting system. The authors point out that differences in training and test images result in counting bias. This means that every time conditions change in a biological experiment, there is a need for reannotating and retraining.

The authors have developed a system where user annotates a part of an image and the system propagates the annotations to the rest of the image and presents the results to the user. Then the user can annotate another part of the image where the system has made mistakes. When the user is satisfied with the results, the system provides the count of the objects for the image.

The authors note that counting by object density estimation has so far been more accurate and faster than counting by object detection. Therefore, they use object density estimation in their system. Counting by density estimation involves learning a mapping between local image features and object density. Integrating over regions of the density map provides an object count for that region. An aspect of density-based counting is that density is not informative for a human user and cannot be used to verify the accuracy of the counting.

The authors make three contributions. They provide a simplified approach for supervised density learning based on ridge regression, because it is faster to train which is necessary for an interactive system. They propose two ways to visualize the estimated density so that users could identify where the system has made mistakes. This allows

the system to incorporate users' feedback. They propose an online code-book learning that re-estimates feature encoding as the user continues annotating.

Counting is conducted in a feedback loop. At each iteration, the user selects part of an image and then dots the objects in that region. Given a set of dotted pixels, the system builds a code-book of low-level features, learns a mapping from entries in the codebook to object density, uses the mapping to estimate object density in the entire image and presents that estimation in an intuitive visualization. By using the visualisation, the user can spot errors and provide further annotations for those regions.

2.5 Counting in the Wild

This thesis attempts to solve the same problem as "Counting in the wild" [3]. This article also describes the background of this problem. During the course of ecological surveys of Antarctic penguins, images are collected automatically with fixed cameras. Images are collected every hour on 40 different sites. Zoologists are interested in the size of the penguin population on each site and how much it changes. This can be studied for correlation with climate change. Therefore, it is necessary to count the penguins in each image. So far this has been done by humans, but this process could be automated.

As the authors of the article point out, the images present us with many difficulties – variability of vantage points of the cameras, variation of penguin scales, weather conditions, similarity between penguins and some of the surrounding objects (for example rocks), and crowding. Different sites of the penguin dataset have different properties. Some cameras capture wide shots with masses of penguins. Other cameras capture constantly occluded images.

The annotation process is conducted on a public website [1]. There, volunteers annotate images by placing dots inside penguins on an image. There have been 35 thousand different annotators. An image is removed from the site after it has been annotated by 20 people. This has resulted in a dataset with a large number of dot-annotated images.

The authors point out that crowd-sourced annotations present us with extra difficulty, because they contain many errors and contradictions. Especially on difficult images, the

annotators often under-count. When an image is cluttered, it is easy for a human to miss an instance. Therefore, it is necessary to build a model that can handle noisy labels.

Dot annotations are an easy way to label images and they are most commonly used in counting tasks. But the problem with dot annotations is that they do not capture object size which varies a lot in the dataset. Also, simple annotations require more complex models.

The authors propose a new approach for learning to count. They extend other density-based methods by incorporating foreground-background segmentation into the learning process and by taking advantage of multiple annotations. There are spatial variations between annotations that can offer cues about object scale. Counting variability between annotations can be used to predict the annotation difficulty. The authors use a deep multi-task network that joins the three components – object-density prediction, foreground-background segmentation, and local uncertainty estimation.

The authors have trained a convolutional neural network to estimate a density function $\lambda(p)$ on a novel image. Integrating over any region of the function $\lambda(p)$ will return the count of the objects in that region. A prediction of the agreement map $u(p)$ can be used to estimate how much multiple annotators would agree on the object count of a region of an image. This also indicates image difficulty.

The authors point out that regressing the object density function from dot annotations requires knowledge of the size of each object. It is necessary to have a depth map with information about the area covered by each object or bounding box annotations. The authors propose a new method for defining the object density map by using object segmentation.

The authors present a multi-task convolutional neural network which produces foreground/background segmentation $s(p)$, an object density function $\lambda(p)$ and a prediction of agreement between the annotators $u(p)$. A fundamental aspect of their framework is the way the labels are defined for different learning tasks. Given a set of dots, they define a trimap of positive, negative, and ignore regions. They correspond to regions contained inside the object, regions of the background, and uncertainty regions in between.

It is necessary to define a regression target for each task. For the segmentation map target, the positive and negative regions define the foreground and background, whereas the ignore regions do not contribute in the computation of the loss. The density map target is obtained from the predicted segmentation and user annotations. Connected components are obtained from predicted segmentation and for each an integer score is assigned as the maximum of different annotators. Density target for each pixel of the connected component is the integer score divided by the component area. The uncertainty map target consists the variance of the annotations within each of the connected components.

The authors train for the three tasks in parallel and end-to-end. They use higher weight for the segmentation loss.

The authors use dot annotations provided by multiple annotators to generate trimap t(p) for that image. Penguins that are further from the camera are smaller and this means that for this dataset penguins get smaller from bottom to the top of the image. In one case the authors assume that there is a depth map for the scene together with an estimated object size. The authors use the following computation. Distance transform is computed from all dot annotations for each pixel. The trimap positive, negative, and ignore regions are obtained by thresholding the distance transform on the predicted object size.

The authors point out that it is difficult to evaluate the results of the experiments because there is no ground truth for the penguin dataset. Also, the counts provided by the annotators are usually a lot smaller than the true count. The authors propose an evaluation metric that reflects the similarity of automatic estimations to the ones provided by the annotators and also their uncertainty. The authors take into account that the annotators typically undercount and propose to compare with the region-wise max of the annotators.

The authors used lower-crowded and medium/lower-crowded sites of the penguin dataset, which add up to about 82 thousand images. They split the images into training and test sets in two different ways – they use both mixed-sites split and separated-sites split. In the first case images from the same camera can appear both in training and test sets, in the second case not. Their training set size is 70% of all the used images.

This is the first work that addresses the problem of counting from crowd-sourced dot-annotations, so the authors have no baseline to compare their work with. They compare their work with density-only baseline.

The authors observe that the error of their methods is bigger with pictures with higher density. But high density also affects annotation error.

The authors examine the effect of the number of annotators on the proposed counting methods. The counting accuracy improves as the number of annotators per image increases.

2.6 YOLO algorithm

The approach of this thesis is based on the YOLO algorithm which is described in [4]. YOLO is an object detection algorithm. It uses single neural network. Full images are the input and the output is bounding boxes and class probabilities of detected objects.

In their paper, the authors also describe existing approaches for object detection. They point out that other detection systems use classifiers for detecting an object. Classifier is used on various scales and locations on the image to see if it contains an object. Some systems use sliding windows approach, others use region proposal methods. The latter approach first generates potential bounding boxes and then runs classifiers on them.

The authors have reframed object detection as single regression problem. Image pixels are translated to bounding boxes and class probabilities. The name of the approach, YOLO, comes from the fact that you only look once at an image to predict its contents. As the authors point out, YOLO is a simple approach, it uses a single convolutional network.

The authors point out the benefits of the YOLO approach. YOLO is fast and more precise than other real-time systems. When making predictions, YOLO reasons globally about the whole image. Unlike other approaches, like sliding windows and region proposal based methods, YOLO looks at the whole image. This means that it encodes contextual information. Another benefit is that YOLO learns generalizable representations.

The authors have unified the separate components of object detection into a single network. For predicting, the network uses features from the entire image and reasons globally about the full image.

The input images are divided into an S by S grid. A grid cell is responsible for detecting an object if the centre of that object is inside that grid cell. Each grid cell predicts B bounding boxes and confidence scores for them. The confidence score represents the probability that the bounding box contains an object and how accurate is the box.

Each bounding box consists of 5 predictions: the x and y coordinates of the centre of the box relative to the grid cell, the width and height of the object and the confidence prediction which represents how accurate is the box.

Each grid cell also predicts C conditional class probabilities. At test time the conditional class probabilities and box confidence predictions are multiplied to get class-specific confidence scores for each box. These scores encode the probability of the class appearing in the box and how accurate is the box.

The model is implemented as a convolutional neural network. In the network the convolutional layers extract features and the fully connected layers predict output probabilities and coordinates. The network architecture is inspired by GoogLeNet model and has 24 convolutional layers followed by 2 fully connected layers.

The model is optimized for sum-squared error, which was chosen because it is easy to optimize, but it has some limitations. There is a problem with grid cells that do not contain any objects. They can overpower the gradient form cells that do contain objects and cause model instability. To remedy this, the loss for boxes that do not contain objects is decreased.

3 Methodology

This chapter describes the dataset that was used for training the models. In the dataset, the annotations were provided as dot annotations. Since YOLO algorithm requires bounding boxes as input, bounding boxes were generated from dot annotations. Different parameters that were used for training the models are described. Since YOLO algorithm outputs bounding boxes, but the ground truth is provided as dot annotations, there is a need for a special evaluation algorithm.

3.1 Dataset

The original dataset is available at [2]. It contains images from different locations. All the images have approximately the same resolution, but the scale of the penguins varies quite a lot. On the images, penguin diameter is between 15 and 700 pixels. In the experiments of this thesis, images where the penguin scale is similar were used. About half of the locations from the original dataset where chosen and images from these where used for the experiments. The reduced dataset contains about 50 000 images.

The original images had a resolution of either 2048 by 1536 pixels or 1920 by 1080 pixels. Training a model with such high-resolution images would have been computationally expensive and would have taken a lot of time. It is questionable if the model would have benefitted from having such high-resolution images as input, because the penguins on these images are large enough so that for the human eye, they are recognizable even if the resolution of the images is reduced.

The images were resized to have a resolution of 448 by 448 pixels. This reduced the resolution of the images, but the penguins are recognizable also on the resulting images. This resizing also resulted in all the images having the same dimensions. Since the original images had different dimensions and they were resized to have the same dimensions, some of the resulting images are a bit distorted. But the distortion is small and should not have a negative effect on the resulting model.

The images were divided into a training set of about 40 000 images and a test set of about 10 000 images. Both the training and test sets contain images from all the locations. For dividing the images into training and test sets a program looped over all the images and put every fifth image into the test set and the rest constituted the training set.

The dataset also contained information about the average size of the penguin on different image locations. This is useful because penguins that are further from the camera appear smaller. This information was presented by a special grey-scale image where each pixel value represented the average penguin size at that pixel location. The information about penguin sizes was used to generate bounding boxes from dot annotations.

The images have been annotated by dot annotations – humans have annotated the images by placing a dot inside a penguin instance. These dot annotations are available as a list of picture coordinates. For each penguin, there should be corresponding coordinates in the list. Each image has been annotated by several annotators, but in the experiments of this thesis, for each image the annotations of only one annotator were used.

Since the dataset was not annotated by experts, but by ordinary people, it contains mistakes. Especially when there is crowding on the image, people have often missed penguin instances. This causes difficulties both when training the model and when evaluating the results. When training the model, the model should handle the fact that some of the instances are not marked. When evaluating the model, we do not have absolute ground truth.

The reduced dataset contains images from different locations, each location is marked with a code name (for example DAMOa). Since the images from different locations have different properties, statistics about different locations is presented:

Table 1 Statistics about the different locations of the penguin dataset

	Maximum	**Minimum**	**Average**
DAMOa	22	0	10.66
GEORa	94	0	11.09
HALFb	73	0	24.59
HALFc	21	8	14.14
LOCKb	33	0	4.90
NEKOc	74	0	14.17
PETEc	96	0	9.82
PETEd	63	0	8.89
PETEf	83	0	12.11
SPIGa	60	0	8.61

Table contains information about the maximum, minimum, and average number of penguins for images from different locations.

3.2 Generating bounding boxes

In the original dataset, penguins were annotated by dot annotations. For each image, several annotators have provided penguin locations by marking each penguin with a dot. For the experiments of this thesis, for each image, the annotations of one annotator were used. In the dataset, for each image there is a list of picture coordinates that represent the locations of the penguins.

The dataset also contains information about the average penguin size at each image location. For each picture, there is a corresponding grey-scale picture where each pixel value represents the average penguin size at that pixel location.

YOLO algorithm requires bounding-box annotations. Using the original dot-annotations and information about average penguin sizes bounding boxes were generated. For the experiments, two sets of bounding boxes were generated, one containing smaller bounding boxes and the other larger.

The following algorithm was used. The program looks at each dot annotation and penguin size at the location of the dot annotation. To generate the smaller bounding boxes one fourth of the average penguin size is used for calculations. To get the y coordinate of the upper left corner of the bounding box, one fourth of the penguin height is subtracted from the y coordinate of the dot annotation. To get the x coordinate of the upper left corner of the bounding box, one fourth of the penguin width is subtracted from the x coordinate of the dot annotation. To get the upper right corner of the bounding box one fourth of the penguin height is subtracted and one fourth of the penguin width is added to the dot annotation coordinates. To get the lower left corner of the bounding box, one fourth of the penguin height is added and one fourth of the penguin width is subtracted from the dot annotation coordinates. To get the lower right corner of the bounding box, one fourth of the penguin height and one fourth of the penguin width is added to the dot annotation coordinates.

For many reasons the generated bounding boxes are not that precise. The original dot annotations are usually not located at the centre of the object. Penguins are not the same size. Some of the penguins are standing up and some are lying down. For some images this algorithm worked better than for others.

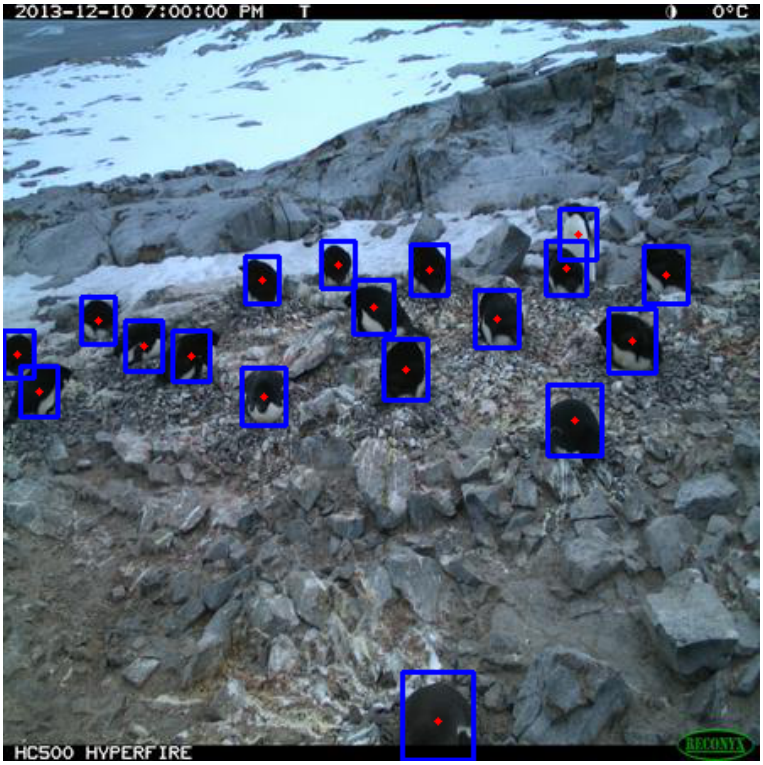

Figure 1 Example of an image where smaller generated bounding boxes work well

Figure 1 represents an example where this algorithm worked quite well. On the image the original dot annotations are marked with red dots and the generated bounding boxes are displayed as blue rectangles. Almost all the penguins are inside the generated bounding box and the bounding boxes are not too big compared to the penguins.

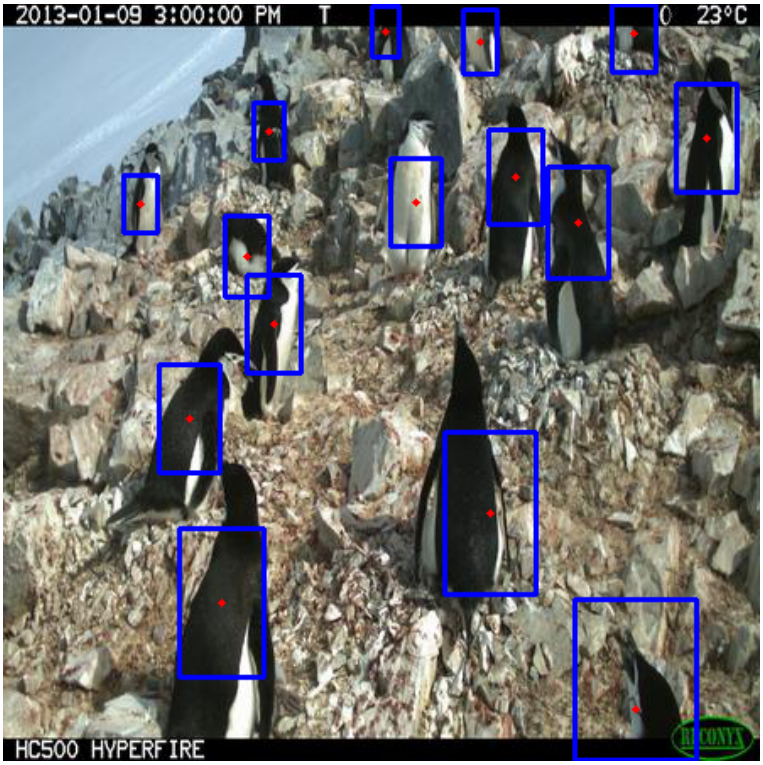

Figure 2 Example of an image where smaller generated bounding boxes do not work well

Figure 2 represents an example where this algorithm was less successful. For the penguins on this image the generated bounding boxes are generally too small – they contain only a part of the penguin.

These smaller bounding boxes often do not contain the whole penguin. Because of that larger bounding boxes were also generated. The same algorithm was used with the difference that one third of the penguin size was used for calculations. With the larger bounding boxes it is more likely that the whole penguin is contained inside the bounding box, but it is also more likely that the bounding box contains background.

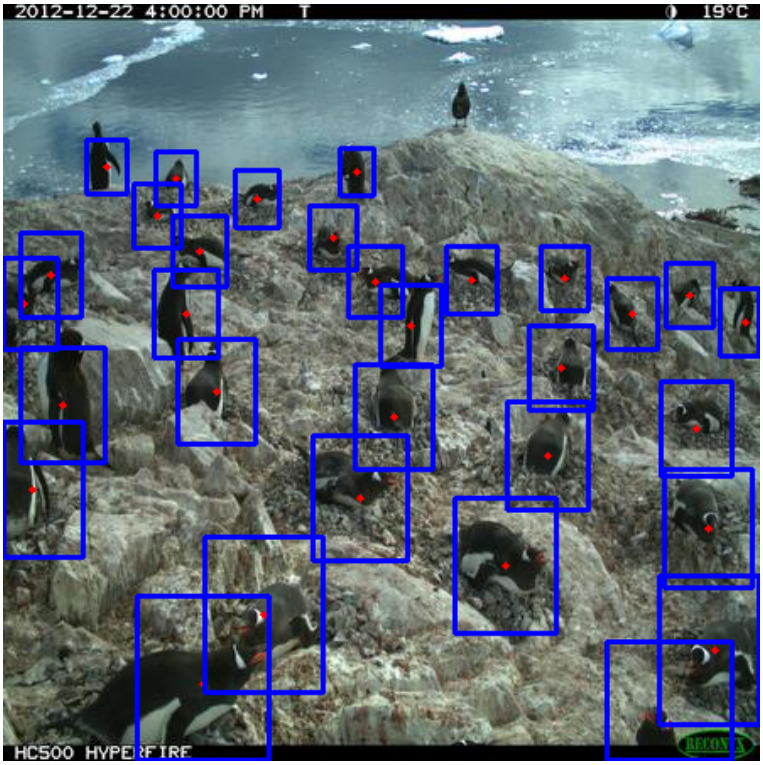

Figure 3 Example of larger generated bounding boxes

Figure 3 represents an example of the larger generated bounding boxes. Most of the penguins are contained inside the generated bounding box. At the same time the bounding boxes also contain some of the background.

3.3 Training the model

The models were trained with an implementation of the YOLO algorithm found at [13]. This is a TensorFlow [14] implementation of the YOLO algorithm. The existing implementation was modified so that it could be used with the penguin dataset. The main modification was in loading the input data while training. Google Cloud [15] was used for training the models.

Different models were trained using the penguin pictures and generated bounding boxes as input. Six different models were trained using different parameters. The models were trained using either smaller or larger generated bounding boxes. The models were trained for different numbers of iterations. One hyper parameter of the YOLO algorithm is the number of boxes per grid cell. Image is divided into grid cells and each grid cell is responsible for detecting a number of bounding boxes specified by this parameter. Since some of the images are quite crowded, different values were used for this parameter.

Other parameters of the algorithm were kept constant. Learning rate of 0.0001 and batch size of 10 were used. Using a batch size on 10 and training for 5000 iterations means that some of the images were used multiple times for training, since training set size was about 40 000.

3.4 Evaluating the results

Since the original dataset was annotated by dot annotations and the YOLO algorithm outputs bounding boxes, it is not trivial to evaluate the performance of a model. It would not be reasonable to evaluate the 10 000 test images by hand. To automate the process of evaluating the results the following algorithm was used.

In the beginning, counters for true positives, false positives and false negatives are all initialized to zero. The program loops over the output bounding boxes. If the bounding box contains a dot annotation, the counter for true positives is incremented by one. Also, the corresponding dot annotation is removed from dot annotations list so that the same dot annotation could not be used twice. If the bounding box does not contain any dot annotations, the counter for false positives is incremented by one. After the program has looped over all the bounding boxes, the count of false negatives is calculated by subtracting true positives count from the number of dot annotations.

This algorithm approximates the performance of a model, but it is not precise as is illustrated by the following example image.

Figure 4 Example of an image where the evaluation algorithm does not work correctly

Figure 4 demonstrates an example where the evaluation algorithm would not work absolutely correctly. One of the bounding boxes contains at least a part of a penguin and should be counted as a true positive, but since the dot annotation is on another part of the penguin, the bounding box is counted as false negative. Still, this algorithm is a way to approximate the performance of a model.

4 Results

After training the models with different parameters, each of them was used with test set images. The evaluation algorithm was used on the outputs to evaluate the performance of each model. This chapter presents the results for each model. Since the results vary depending on image locations, the results for different locations are also presented.

4.1 Results of different models

To evaluate the results, for each image, I used the annotations of one annotator. Based on the annotations I used there are all together 101 226 penguins on test set images. The following table presents the results for different models. For each model are presented:

- Whether smaller or larger generated bounding boxes were used while training the model.
- For how many iterations the model was trained.
- Boxes per grid cell – this is a hyper parameter of the YOLO algorithm.
- Threshold – YOLO algorithm outputs a confidence score for each bounding box. This can be used as a threshold when filtering the results.
- Intersection over union threshold – another threshold that can be used when filtering the results.
- True positives – how many penguins did the model identify correctly on all the images together.
- False positives – how many false detections the model generated for all the images together.
- False negatives – how many penguins did the model miss for all the images together.

- Precision – calculated based on global true positives and false positives.
- Recall – calculated based on global true positives and false negatives.
- F1 score – calculated based on the previous precision and recall.
- Average precision of images – the average of the precisions calculated for each image.
- Average recall of images – the average of the recalls calculated for each image.
- Average F1 score of images – the average of the F1 scores calculated for each image.

Table 2 Results of different models

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
Smaller/larger bounding boxes	Smaller	Larger	Smaller	Larger	Smaller	Larger	Larger
Number of iterations	5 000	5 000	5 000	5 000	10 000	10 000	10 000
Boxes per grid cell	2	2	4	4	2	2	2
Threshold	0.2	0.2	0.05	0.05	0.2	0.2	0.1
IOU threshold	0.4	0.4	0.1	0.1	0.4	0.4	0.4
True positives	4 913	12 237	4 906	15 492	5 262	15 951	36 127
False positives	3 364	5 683	33 484	64 648	2 233	7 193	42 115
False negatives	96 313	88 989	96 320	85 734	95 964	85 275	65 099
Precision	0.59	0.68	0.13	0.19	0.70	0.69	0.46
Recall	0.05	0.12	0.05	0.15	0.05	0.16	0.36
F1 score	0.09	0.21	0.07	0.17	0.10	0.26	0.40
Average precision of images	0.46	0.61	0.15	0.21	0.53	0.63	0.44
Average recall of images	0.06	0.14	0.04	0.14	0.06	0.17	0.37
Average F1 score of images	0.10	0.21	0.06	0.16	0.11	0.26	0.39

For all the models, the main problem seems to be the large number of false negatives – meaning that all the models miss a lot of the penguin instances. The YOLO algorithm outputs a confidence score for each bounding box and the bounding boxes can be filtered based on that. Model 7 is the same as model 6, but with a lower threshold for

filtering the results. Model 7 outputs more bounding boxes and is able to detect more penguins, but the number of false positives also increases – background is identified as penguin more often.

Model 7 is performing the best. Models 5, 6 and 7 use the same parameters as models 1 and 2, but they were trained for more iterations – some of the training images were used multiple times while training. Although they are performing better than the other models, it would not be reasonable to train for even more iterations, because using the same images for training will eventually lead to overfitting.

Figure 5 Example of model 6 performance

Figure 5 exemplifies model 6 performance. On the images in this chapter, the red dots represent ground truth dot annotations and the blue rectangles represent the bounding boxes that the model output. On this image, the model is able to recognise some of the penguins. The model is not identifying background as penguins, but it is missing a lot of penguin instances.

Figure 6 Example of model 7 performance

Figure 6 exemplifies model 7 performance. Model 7 is the same as model 6, but the threshold for filtering the results is lowered to 0.1. The model is able to recognize more penguins. There are more false positives. There seems to be a problem with overlapping bounding boxes – in some cases there are multiple bounding boxes for the same bird.

Model 3 is performing worst. Using the initial threshold of 0.2 this model detected almost no objects. When the threshold was lowered, the model was able to detect some objects, but it is also generating a lot of false positives.

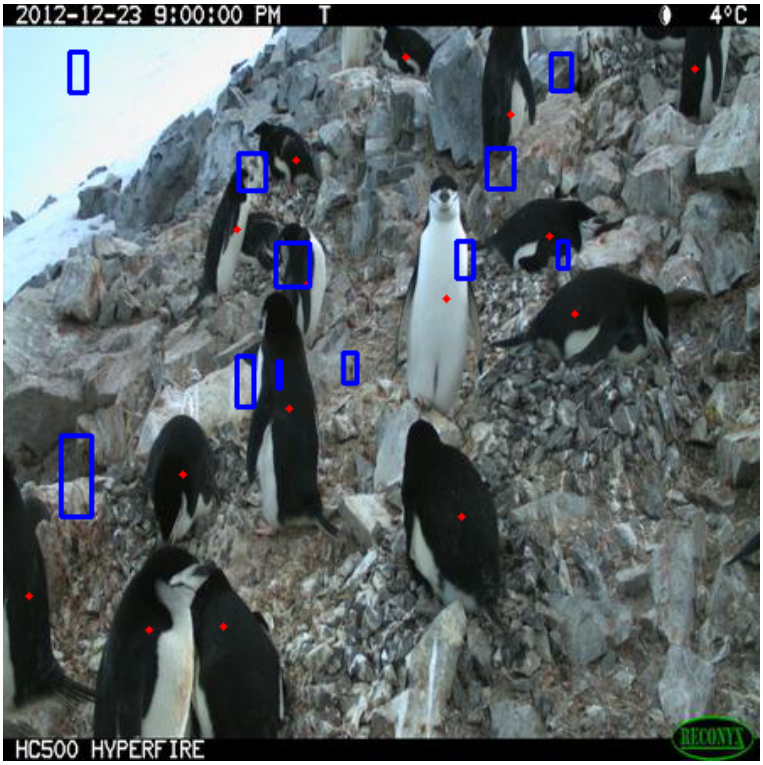

Figure 7 Example of model 3 performance

Figure 7 exemplifies model 3 performance. The bounding boxes that the model outputs are a lot smaller. The model is detecting almost none of the penguins. The model is identifying background as penguins quite a lot.

The results of the three best performing models are presented in more detail.

4.2 Results of model 2

The original pictures were taken at different locations which are represented by code names (for example DAMOa). Different locations have different properties – average penguin size, perspective, crowding of the birds. Since all these properties influence the performance of the model, it is useful to look at the results of all the locations separately. The following table represents the results of model 2 for different locations:

Table 3 Results of model 2

	True positives	False positives	False negatives	Precision	Recall	F1 score
DAMOa	195	58	688	0.77	0.22	0.34
GEORa	1133	698	8815	0.62	0.11	0.19
HALFb	261	61	1267	0.81	0.17	0.28
HALFc	232	49	573	0.83	0.29	0.43
LOCKb	1251	396	3833	0.76	0.25	0.38
NEKOc	1587	615	12154	0.72	0.12	0.21
PETEc	2754	1513	20519	0.65	0.12	0.20
PETEd	2112	1133	15791	0.65	0.12	0.20
PETEf	1033	646	17081	0.62	0.06	0.11
SPIGa	1679	514	8268	0.77	0.17	0.29

The F1 score for the location with the best performance is 0.43 and for the location with the worst performance 0.11, which is a significant difference. To understand why the results vary so much, it is useful to look at example images from these locations.

Figure 8 Example image from location HALFc

Figure 8 represents an example from location codenamed HALFc, which is the location where model 2 had the best performance. At this location, the penguins are quite large compared to the picture size. There is not much crowding between the birds. The contrast between the birds and the background is quite high.

Figure 9 Example image from location PETEf

Figure 9 represents an example from location codenamed PETEf, which is the location where model 2 had the worst performance. On this image penguins are smaller compared to the picture size. There is more crowding between the birds.

4.3 Results of model 6

The following table presents the results for different locations for model 6.

Table 4 Results of model 6

	True positives	**False positives**	**False negatives**	**Precision**	**Recall**	**F1 score**
DAMOa	246	94	637	0.72	0.28	0.40
GEORa	1391	868	8557	0.62	0.14	0.23
HALFb	303	81	1225	0.79	0.20	0.32
HALFc	270	62	535	0.81	0.34	0.47

LOCKb	1455	492	3629	0.75	0.29	0.41
NEKOc	2215	947	11 526	0.70	0.16	0.26
PETEc	3652	1796	19 621	0.67	0.16	0.25
PETEd	2865	1347	15 038	0.68	0.16	0.26
PETEf	1451	767	16 663	0.65	0.08	0.14
SPIGa	2103	739	7844	0.74	0.21	0.33

Compared with model 2, recall has improved more, whilst precision has gotten worse for some locations. The F1 scores have improved. The location where the model has the best performance and the location where the model has the worst performance are the same as for model 2.

4.4 Results of model 7

Model 7 is the same as model 6, but with a lower threshold for filtering the results. The following table presents the results of model 7 for different locations:

Table 5 Results of model 7

	True positives	**False positives**	**False negatives**	**Precision**	**Recall**	**F1 score**
DAMOa	450	562	433	0.44	0.51	0.47
GEORa	2908	4072	7040	0.42	0.29	0.34
HALFb	479	786	1049	0.38	0.31	0.34
HALFc	444	517	361	0.46	0.55	0.50
LOCKb	2477	3375	2607	0.42	0.49	0.45
NEKOc	4842	6342	8899	0.43	0.35	0.39

PETEc	9235	9484	14 038	0.49	0.40	0.44
PETEd	7305	7012	10 598	0.51	0.41	0.45
PETEf	4050	4742	14 064	0.46	0.22	0.30
SPIGa	3937	5223	6010	0.43	0.40	0.41

With lower threshold recall improves – the model is able to correctly detect more penguins. The precision has gotten worse – the model is generating more false positives. The F1 scores have gotten better.

5 Discussion

The goal of this thesis was to count objects in images, specifically to count penguins in images taken in the course of a survey of Antarctic penguins. The images were taken periodically with fixed cameras at different locations. Based on these images, scientists want to monitor penguin populations and how they change over time. Therefore, it is necessary to count the penguins on these images.

So far, this has been done by ordinary people on a public website. There, anybody can join the project and process the images by placing dots on penguin instances. This has resulted in a large dataset of dot-annotated images, which can be used to train machine learning models that would automate the process of counting penguins in images.

This thesis is not the first attempt to count penguins in these images. “Counting in the wild” [3] deals with the same dataset for the same purpose. The approach used in this thesis is different from the approach used in “Counting in the wild”. In general, there are two different ways to count objects in images – one based on object detection and the other based on object density estimation. “Counting in the wild” is based on density estimation – their model learns a mapping between image features and density. Integrating this function over an image region, gives object count for that region. The approach used in this thesis is based on object detection.

In this thesis, the YOLO algorithm is used to detect penguins in images. This algorithm is different from previous detection methods, in that it does not use sliding windows or region-proposal methods to detect objects. Instead it looks at the whole image and tries to detect all the objects in the image in a single go. Image is divided into grid cells and each grid cell is responsible for detecting the object if the centre of that object is inside that grid cell. A model trained with the YOLO algorithm outputs bounding boxes and confidence scores of detected objects.

The penguin dataset was annotated with dot-annotations. The YOLO algorithm requires input as bounding boxes. This was solved by generating bounding boxes from dot

annotations. The dataset also contained information about average penguin size for each picture location. A fraction of the average penguin size was subtracted or added to the coordinates of the dot annotations to get the coordinates of the four corners of the bounding box. Bounding boxes of two different sizes were generated to compare their performance.

Since the YOLO algorithm outputs bounding boxes, but the ground truth was provided as dot annotations, there was also a need for a special evaluation algorithm to evaluate the performance of the trained models. In general, the evaluation algorithm counts the bounding boxes that contain a dot annotation as true positives and the bounding boxes that do not contain a dot annotation as false positives.

Using an existing implementation of the YOLO algorithm, the penguin dataset, and the generated bounding boxes, seven different models were trained with different parameters.

The models that were trained with the larger generated bounding boxes performed better on the test set. The difference with the smaller and larger generated bounding boxes is that the smaller bounding boxes often contained only a part of the penguin. At the same time the smaller generated bounding boxes had the benefit that they did not contain much background information. With the larger generated bounding boxes, it is more likely that the whole penguin is contained inside the bounding box. At the same time, the larger bounding boxes contained more background information.

The smaller generated bounding boxes could be thought of as part detectors – the model learns to detect different parts of the penguin. This presents a problem at test time – given a new image the model would identify different parts of the penguin on that. The same bird would be detected multiple times. Ideally, different detections of the same bird should be merged into one – but this is a difficult problem to solve.

The reason why smaller generated bounding boxes performed worse is probably that the training set did not contain enough data. Since the smaller generated bounding boxes often contained only part of a penguin, the model was learning to detect different parts of the penguin. When learning to detect different parts of the penguin and when for each penguin only one part of it is annotated, more data is needed than when learning to

detect whole penguins. This is probably the reason why the models that were trained with larger generated bounding boxes performed better.

The images in the dataset were taken at different locations. Images from different locations have different properties – average penguin size on the image, average penguin count on the image, crowding and inter-occlusion between birds. The models performed better on images where penguins appeared quite large compared to the picture size. The models performed better on images where there was not much crowding between the birds. One indicator of how much crowding there is for a specific location, is the maximum number of penguins at that location. When the maximum number of penguins is high for a location, that means that it is more likely that there is crowding on images taken at that location. Models performed worse for locations where there is more crowding. This is in accordance with other research on counting objects in images. In general, detection-based methods do not perform well, when objects are overlapping and there is inter-occlusion between them.

It is not possible to directly compare the results achieved in this thesis and the results of "Counting in the wild". Density-based methods output plain object counts, which can be compared with ground truth counts. "Counting in the wild" uses mean counting error as the evaluation metric. The models of this thesis output object locations and when evaluating the results, it would not be correct to count the output bounding boxes and compare that with ground truth counts. Since whether the bounding box contains an object must also be taken into account.

It seems that density estimation based method is more suitable for the penguin dataset. Detection-based methods perform better when there is not much overlapping or crowding between instances. The approach used in this thesis worked better for images from locations where there is not much crowding. But most of the images of the penguin dataset are crowded. On the images birds are usually grouped together. In such cases, density estimation based methods work better, because they do not aim to detect individual instances, instead they look at image structure.

The approach used in this thesis worked better for images where penguins were quite large compared to the image size. When the object is small on the image, it probably does not have enough features for an object detector to recognize it. Image resolution

was reduced for the experiments, because using the original images would have been computationally expensive. It is possible that when the YOLO algorithm were trained with the images with the original resolution, it would have performed better.

6 Summary

As part of a survey of Antarctic penguins, images are taken periodically with fixed cameras. Zoologists are interested in the sizes of penguin colonies. Therefore, there is a need to count the penguins in these images. So far this has been done by ordinary people on a public website. This has resulted in a large dataset of dot-annotated images that can be used to train and test machine learning models.

In general, there are two approaches to count objects in images. One is based on density estimation and the other is based on object detection. “Counting in the wild” attempts to count the penguins using a method that is based on density estimation. This thesis presents a way to count the penguins using a method that is based on object detection.

Based on dot annotations of the dataset, bounding boxes were generated. The YOLO algorithm was used to train different models using the penguin dataset with the generated bounding boxes. An algorithm for evaluating the results was presented. Results of different models were presented.

References

[1] “Penguin watch,” [Online]. Available: https://www.penguinwatch.org. [Accessed May 2018].

[2] “Penguins dataset,” [Online]. Available: http://www.robots.ox.ac.uk/~vgg/research/penguins/. [Accessed May 2018].

[3] C. Arteta, V. Lempitsky and A. Zisserman, “Counting in the wild,” *European Conference on Computer Vision,* pp. 483-498, 2016.

[4] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 779-788, 2016.

[5] V. Lempitsky and A. Zisserman, “Learning to count objects in images,” *Advances in Neural Information Processing Systems,* pp. 1324-1332, 2010.

[6] C. L. V. N. J. Z. A. Arteta, “Learning to detect partially overlapping instances,” *Computer Vision and Pattern Recognition (CVPR),* pp. 3230-3237, 2013.

[7] B. Wu, N. Ram and Y. Li, “Segmentation of multiple, partially occluded objects by grouping, merging, assigning part detection responses,” *Computer Vision and Pattern Recognition,* pp. 1-8, 2008.

[8] D. Kong, D. Gray and H. Tao, “A viewpoint invariant approach for crowd counting,” *Intl. Conf. Pattern Recognition,* pp. 1187-1190, 2006.

[9] W. Xie, J. A. Noble and A. Zisserman, “Microscopy cell counting and detection with fully convolutional regression networks,” *Computer methods in biomechanics and biomedical engineering: Imaging & Visualization,* pp. 283-292, 2018.

[10] Z. Ma, L. Yu and A. B. Chan, “Small instance detection by integer programming on object density maps,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* pp. 3689-3697, 2015.

[11] C. Zhang, H. Li, X. Wang and X. Yang, “Cross-scene crowd counting via deep convolutional neural networks,” *Computer Vision and Pattern Recognition (CVPR),* pp. 833-841, 2015.

[12] A. Carlos, V. Lempitsky, J. A. Noble and A. Zisserman, “Interactive object counting,” *European Conference on Computer Vision,* pp. 504-518, 2014.

[13] “yolo_tensorflow,” [Online]. Available: https://github.com/hizhangp/yolo_tensorflow. [Accessed May 2018].

[14] “TensorFlow,” [Online]. Available: https://www.tensorflow.org/. [Accessed May 2018].

[15] “Google Cloud,” [Online]. Available: https://cloud.google.com/. [Accessed May 2018].

Appendix 1 – Source code of the models

Git repository:

https://github.com/liisharjo/yolo_tensorflow