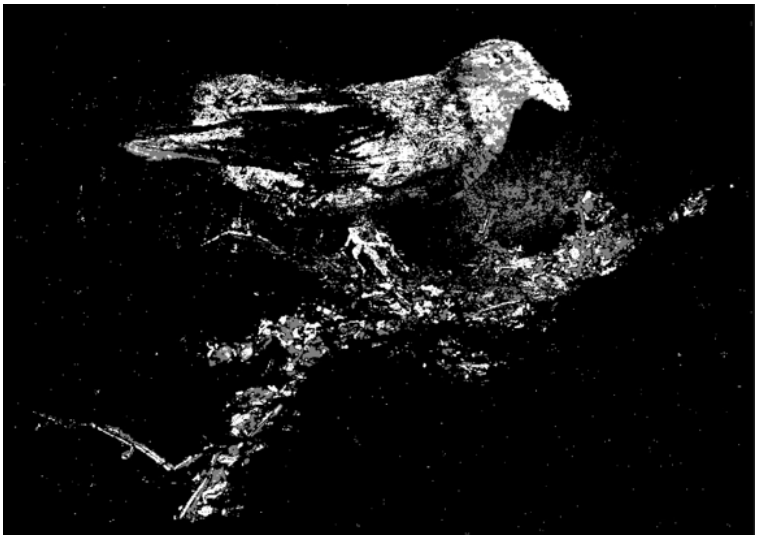
EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science

Lightweight wildlife classification on the edge

Master Thesis

Jakob Laurens Johannes Maria Serlier



Supervisor:
Prof. Dr. Nirvana Meratnia

Eindhoven, April 2023

Abstract

Wildlife camera trapping is crucial for directing biodiversity conservation efforts around the globe. In this thesis, we investigated which AI model and data design choices increase the viability of running wildlife classification models on the edge. We explored which input image dimensionality reduction techniques can be used to decrease the storage and networking requirements on the edge and assist in reducing model size. These techniques include gray-scale conversion, image resolution reduction, compression, and reducing the number of training images. Using Yolov5s as a benchmark model, we show that it is possible to greatly reduce the input dimensions and image size using a combination of these techniques without much compromise in terms of classification performance. In the most constrained input, the performance of the benchmark model is significantly reduced. We propose a novel VAE-SVM model approach tailored for input images with greatly reduced dimensions. The VAE-SVM model outperforms Yolov5s in the most constrained subsets, in particular for small animals such as rodents, while the VAE-SVM model size is significantly smaller compared to the benchmark. Finally, we assessed alternative and non-traditional techniques to aid in wildlife classification on the edge. These techniques include the use of color histograms and local binary patterns, as well as a sequencing trick that uses background removal and cropping around the moving animal. The first two approaches did not yield any significant performance improvements, but the sequencing trick showed promise as a viable technique to improve the performance of wildlife classification in specific cases. These findings provide valuable insights and contribute to the development and feasibility of wildlife classification on the edge.

Preface

This thesis marks the end of my academic journey both at Eindhoven University of Technology and as a student in general. My bachelor's degree was quite different from a Master's in Computer Science: as a Systems Engineering, Policy Analysis, and Management student from TU Delft, I was given the impression that anything which is too technical should be avoided, and complex tasks deserve a meeting and a presentation. Thankfully, while on exchange at Carnegie Mellon University, I got the chance to work on complex engineering tasks and work with people who excel at it. This opportunity gave me the realization that I was still ready to take on a more challenging degree. After becoming an Ivy League dropout due to an unnamed 2020 global event, I jumped at the opportunity to continue my academic career at TU Eindhoven, the University that I grew up so close to. This thesis not only marks the end but also marks the culmination of my academic and technical development after seven years of studying. Consequently, I am proud, grateful, and happy to present to you this document as the materialization of the work that went into it.

What I will never forget is the support I received before, throughout, and during the delivery of this thesis. First and foremost, I thank my supervisor, Prof. Dr. Nirvana Meratnia, who encouraged, guided, and supported me throughout the journey of working on this thesis. This work is made possible by her guidance and mentorship, our discussions, and her valuable suggestions. I would also like to thank the members of the thesis committee, Dr. Tanir Ozcelebi and Dr. Savio Sciancalepore.

Finally, I express my deepest thanks to my friends and family. I thank my friends for supporting me when juggling my work and studies was tough, I thank my girlfriend for all her unconditional love, and I thank my parents and mother especially, who set me on the path of pursuing a Masters's degree, and whose love, wisdom and encouragements have made me the person I am today.

So long and thanks for all the fish,
Jakob

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Keeping track of the number and type of species of wildlife animals around the world using camera traps is an ongoing and important effort. As wildlife counts and diversity have been declining rapidly [16], accurately keeping track of species counts and diversity is crucial for directing conservation efforts. The change and development of species population over time is a warning indicator for the general health of ecosystems [43]. There is an urgent need for global assessment of wildlife diversity, species counts, behavior, distribution, and overall abundance of wildlife [24]. In order to support wildlife conservation efforts, having access to up-to-date indicators such as species count is key in a data-driven approach to address the biodiversity crisis [32]. For the purpose of collecting vast amounts of data for global and regional conservation initiatives and studies, remote monitoring tools have become a popular choice. In particular, there has been a rise in the deployment of camera traps. The popularity of camera trapping stems from their non-invasive nature and the option for standardized data sharing among ecologists [37]. Other remote sensing techniques include drones, or unmanned aerial vehicle (UAV) surveying, and satellites. In this research, we address open challenges in wildlife research in the scope of camera trapping.

Ecology and wildlife research has entered the big data domain, and if it is to scale, automation is required with the support of machine learning in combination with expert domain knowledge [54]. In the context of captured camera trapping imagery, there is a mismatch between the amount of raw (i.e., unlabeled) image data and the resources available to manually label them [54]. Labeling images is a labor-intensive task. Furthermore, the identification of wildlife on camera trap imagery by humans can be prone to errors and systematic population overestimation [29]. As a response to this challenge, the use of machine learning techniques such as Artificial Intelligence (AI), and Deep learning (DL) more specifically, have proven to be useful in the context of wildlife classification, aiding with large-batch image processing. These Deep learning models can learn from existing labeled data (supervised models) and in certain cases from unlabeled data (unsupervised models) [46][41][22][23]. There exist a number of publicly available pre-trained models which are able to classify wildlife images, often with accuracy in the high 90-percentile ranges [11]. The application of custom- and publicly available models for the purpose of classifying wildlife imagery in camera traps have been a significant step forward in allowing wildlife researchers and ecologists to consume and analyze the available data efficiently.

A particular area of interest in the wildlife camera trapping domain is that of deep learning models on the edge. From both a practical and research perspective, there exist an array of challenges to be solved before the theoretical feasibility of edge computing is met with practical use [64]. Even still, classifying wildlife on the edge, meaning: on or near the camera traps, has the potential to address significant challenges which arise in camera trapping. A large portion of available commercial camera trap solutions are manually deployed in the wild and require additional trips on-location for retrieval of the captured images [40]. As these trips can be costly, it can take months before data is collected and ready for analysis. A consequence is that population estimates

following camera trap data measurements are a lagging indicator. Furthermore, allowing classification or pre-processing of images (such as removing empty images) on the edge is beneficial as resources such as energy and storage are scarce on the edge. For instance, if the edge device has connectivity, bandwidth can be reduced by only sending a selection of images to central servers. In terms of storage, if a subset of images can be discarded on the edge, camera traps can be left unsupervised for longer. Consequently, we are interested in increasing the viability of wildlife classification on the edge using deep learning. In the following section, we will discuss some of the challenges of camera trapping and running lightweight AI models on the edge, and further scope our research direction.

Figure 1.1: Two examples of camera trap images containing wildlife. The image on the left is a crow, taken from the ENA24 dataset [61] and the image on the right is displaying a fox, taken from the Channel Islands dataset. [18]

1.1 Problem setting

Classification of camera trap images comes with a set of challenges both technical and non-technical in nature. From a technical perspective, camera traps predominantly have limited options for real-time data transmission [64]. As there is often limited or no internet capabilities on the edge, there can be a significant delay between the monitoring of an animal, and the availability of the data to observers and wildlife researchers. An extension of no real-time data transmission is the lack of general remote access to camera data [64] requiring the need for manual trips to retrieve camera trap data. The proposition of running classification models on the edge can help reduce the frequency of required trips to camera traps. In no-connectivity settings, by filtering out images, which are not interesting to researchers, on the edge, the amount of data that can be stored on the device can be increased. In settings, in which there is sparse connectivity either due to limited network availability or due to energy constraints, camera trap encounters can be caught early or in real time and transmitted for analysis. This can be especially critical in settings such as anti-poaching applications where response speed is of importance [53].

The camera device introduces additional challenges. When a camera trap is triggered it will capture one or multiple images in sequence. When the triggering of the trap is caused by an event that is not an animal, such as an environmental event as moving leaves, empty or "Ghost" images are captured [64]. Ghost images are a significant issue in the labeling and analysis of camera trap images, as they can further increase the labor required to manually label the images as well as influence the classification error rate [29]. Other camera-trapping image-specific challenges include occlusion, motion blur, variance in lighting, and variance in scale [13]. The aforementioned challenges are domain-specific, and we will naturally address these in the scope of this research as a result of running experiments, meaning we will not address any of these challenges in an isolated

setting.

From the deep learning model perspective, the data used for training the models is often label-sparse [46][35]. This means that if a supervised learning approach is used, manual labeling is often required. Generalization of an existing model to a novel environment is not always feasible [11][35], and for species that are relatively rare, generalization of performant models for these species can be especially difficult. Furthermore, data can be further skewed by multiple images taken in the same time frame, meaning that relative animal frequency can further skew the class-wise availability [35]. Fortunately for researchers, a selection of publicly available wildlife models exist which have shown to perform well at detecting animals even in before unseen settings, and as such can aid in efficiently labeling new data. An example of such a model is MegaDetector [11].

Considering the edge domain, the predominant challenges are found throughout existing applications of deep learning on the edge (or *the intelligent edge*). Many of the edge-related challenges found in the Wildlife classification setting are shared across domains and industries. Inference latency of models on Edge can be high [19], energy is sparse on the edge: in the camera trapping setting, certain camera traps run on solar cells as their energy source. Devices capable of running deep learning on the edge require non-trivial computational power. This translates to the issue that Wildlife camera traps are prone to be stolen since they are valuable pieces of equipment. This is especially prevalent in areas where poaching is common as the data can be used to locate specific animals [40]. Updating and maintaining models on the edge can be a complex task, especially when remote or even physical access to the device is limited. These challenges must be addressed both in the academic scope as well as in practical settings in order to ensure successful implementations of deep learning models for Wildlife classification on the edge. Our academic contributions will lie mostly with the challenges introduced in this paragraph.

In this section, we have highlighted challenges that arise in the domain of classifying wildlife on the edge. To reiterate our solution scope, the application of deep learning models has proven to be a key piece in solving the principal problem of dealing with a large amount of raw wildlife imagery and limited (human) resources [60][15][62][22][23][41]. As an isolated solution, deep learning techniques are key in addressing the data and resources mismatch, and by broadening the scope we find that running variants of these models on the edge is the next frontier. Applying deep learning on the edge is a promising approach that requires more research to address the current limitations and challenges that exist in this domain [64]. Given this problem scope, we will now define research questions, which will address both gaps in existing research as well as address open practical challenges.

1.2 Research questions

In the previous sections, we have introduced the problem setting of wildlife classification in the edge domain and have discussed open challenges. The key hypothesis in this study is that the viability of edge AI models running on or near camera traps will increase in the coming years. This increase in practicability and viability will stem from jumps in research on classification models for edge settings, as well as improvements in hardware and processing techniques. The goal of this research is to aid in this aspect, and find methods that further increase this viability. The contributions of this research thereby lie in exploring novel approaches and extending the body of existing research on the topic of wildlife classification on the edge. This results in our main research question:

Which AI model and data design choices can enable edge-based wildlife image classification?

To answer this research question, we formulate the following sub-questions:

I: *Which design choices in reducing input image dimensionality can enable edge-based animal monitoring using AI on the edge?*

The reasoning behind choosing this subquestion is as follows. Images taken on wildlife camera traps are often high in resolution and in RGB. We will explore reducing the input dimensions of these images and the impact they will have on the performance of existing models. By reducing the input dimensions, smaller models, which can be more energy efficient can be used for classification. In addition, the bandwidth and storage requirements of the edge devices can be reduced.

II: *How can we design wildlife classification edge models tailored for reduced input image dimensions?*

Given the reduced input resulting from the previous sub-question, we are interested in designing a model specifically able to cope with constrained data. While traditional models are not specifically designed with highly constrained input dimensionality in mind [34], we look to design a model with an architecture specifically able to deal with constrained input.

III: *To what extent can alternative and non-traditional imaging techniques increase the classification performance of edge models?*

The last sub-question lies in its focus on alternative or non-traditional techniques to increase the viability of wildlife classification on the edge. We will focus on techniques that can be applied during the initial processing of images and may offer unique benefits over just classification using deep learning models.

1.3 Outline

In this chapter, the problem context and research questions have been introduced. In Chapter 2, existing literature on lightweight AI models, and classification of wildlife are reviewed. Chapter 3 presents background on key techniques and models used in answering our research questions. The Methodology Chapter (Chapter 4) describes the specific methods used to answer our questions. The Methodology Chapter also introduces our main contribution, the VAE-SVM model. The experimental setup in Chapter 5 details the dataset selection and cleansing as well as specific implementation details of the experiments. The results of our experiments are described in Chapter 6 followed by the answering of the research questions and conclusions in Chapter 7.

Chapter 2

Literature review

In this chapter, we examine relevant literature on deep learning for wildlife classification purposes. We start by reviewing the current state of research on deep learning for wildlife classification, followed by scoping down on deep learning for wildlife classification on the edge. We conclude the chapter by relating the existing research to our objectives and contributions.

2.1 Deep learning for wildlife classification

As deep learning techniques for image classification matured, these techniques were naturally applied to the domain of wildlife species classification problems as well. As a first, in 2014, Chen Et al. [17] introduced a novel CNN (Convolutional Neural Network) to be used as a classification technique for wildlife specifies recognition. The dataset consisted of 14,346 training images and 9,530 testing images from 20 species from North America. With a reported accuracy of 34 to 38%, it is clear that much ground is still to be covered in deep learning for wildlife classification purposes at that point in time. More recent studies show it to be more viable for practice with performance in the high 90% accuracy range in various datasets and settings.

An array of existing researchers have listed and detailed previous research on wildlife classification using deep learning in depth. For example, Zualkernan et al. performed an extensive literature review in 2022 both in terms of general applications of CNN for animal classification and for the edge specifically [64]. Palanisamy and Ratnarajah perform a systematic review of deep learning for the detection of Wildlife in their 2021 study where they address recent advancements and contributions in the field [49]. These two studies exemplify the abundance of research on the application of current and past state-of-the-art CNNs such as ResNet, VGG, AlexNet, and InceptionNet, either as the backbone for custom models or as the main model for Wildlife classification and detection purposes. What many of the wildlife classification studies have in common is a level of accuracy ranging from the mid-80 % to the high 90% ranges. Examples are Yousif et al. [62], Gomez et al. [23] and Okafor et al. [48] using AlexNet. Miao et al. [41], Nguyen et al. [44], and Gomez et al. ([22] & [23]) using ResNet variants. More recently in a 2023 preprint, Brookes et al. [15] use ResNet-50 as a backbone for a study on Deep Metric Learning for great ape behavioral actions. VGG is used by Yin Z. and You F. [60], and Norouzzadeh et al. [45]. InceptionNet and variants are used in studies by Okafor et al. [48] and Allken et al. [7]. Each of these studies focuses on animal classification, with dataset sizes ranging from low 4-digit numbers to datasets containing mid-seven-digit samples. The diversity of available research and their performance demonstrates the maturity of the domain of wildlife classification using CNNs.

Recent wildlife research efforts have also focused on an active learning approach supporting the labeling and counting of animals in image data [13][46][9][51]. In active learning, the model can request a human expert to label certain unlabeled data points. The goal in this is to only

request manual labeling for data points which will introduce most new information (i.e., improve the model performance the greatest). Thereby, the advantages of active learning are that the total amount of required labeled data can remain low, and biases in the model can be addressed by human interaction with the model. In an article by Miao et al. [42], they propose a hybrid approach to wildlife image classification. In this approach, they combine machine learning and human annotation in an iterative approach which includes a model inference step, a human annotation step by means of active learning, and a model update step. This approach achieves 90% while only requiring 20% of data to be annotated by humans compared to existing approaches. Although an active learning approach is promising for the field as it could result in fewer data that need to be labeled for model training, the approach may not be applicable to support our research objectives.

Given the existing research on wildlife classification, it is important to make a distinction between the type of models used. Processing images in the wildlife domain for species recognition or counting can reasonably be divided into two categories, direct classification or two (or multi)-stage classification. This division of categories has previously been made in research such as that of Kaur and Singh [31], and that of Cuhna et al. [19]. The studies mentioned in the previous paragraph fall into either of the approaches. The direct classification works by classifying an image directly, without prior object detection, examples of this are the works of Cuhna et al. [19] for filtering ghost images on embedded devices, and many of the previously mentioned studies using state-of-the-art CNN-based architectures [22][23][45][48][44]. On the other hand, two-stage classification first detects an object in the image and then assigns a species to the detected animal. An advantage of first detecting an object (i.e., the animal) in an image is that it can be used to preemptively remove empty or 'ghost' images. As removing empty images is such an important factor in practical camera trapping applications and research, not all wildlife research aims to classify species, rather just the detection of the (lack of) animals could be sufficient. For example, Cuhna et al. [19] focus their research to just removing the empty or ghost images.

Outside of traditional academic settings, there exist a variety of open and commercial models specifically designed for wildlife classification. Wildlife researcher Dan Morris, who is involved with a variety of projects relating to DL and camera traps including MegaDetector [11] and Wildlife Insights [6], keeps track of a curated list of camera trap platforms and models using Machine Learning as part of their tooling. A selection of these models includes Camelot [2], an open-source camera trapping software; Wildlife Insights [6] which is a platform that provides tools and techniques for camera trapping purposes; WildEye TrapTagger [5] which is a web application capable of detecting, classifying and counting wildlife; and animl-frontend [3], an open-source frontend for viewing and labeling camera trap data. The team at Microsoft AI for Earth developed a model named MegaDetector which aims to detect objects in camera trap imagery [11]. The model purposefully generalizes to many different environments and settings, so that researchers can be aided in efficiently analyzing their data. MegaDetectorv5 uses the object detection model YOLOv5x6 as its backbone and is therefore particularly interesting as the Yolo (You Only Look Once) family of models is used for benchmarking purposes in this research.

2.1.1 Wildlife classification on the edge

Applying deep learning models on the edge is a more niche subject of research, due in part due to the aforementioned constraints of running models on the edge in the introduction. These include associated costs and risk of hardware, energy constraints; and moreover the current impracticability in real-life scenarios. As remote networking capabilities are improving rapidly, and advancements are made rapidly both in terms of hardware for AI on the edge as well as edge models, the feasibility of running classification on the edge increases. For instance, in recent years, the cost of using Non-Terrestrial Network (NTNs) such as those based on satellite connectivity has decreased significantly [10], which allow for network connectivity in remote areas. The collection

of these advancements in edge computing and its neighboring technologies have led to an increase in interest, viability, and research on wildlife classification on the edge in recent years. In this research, we will not be describing recent advancements in the overlapping fields of edge computing and deep learning specifically, but rather focus on the application in the Wildlife domain. Existing research such as *Convergence of Edge Computing and Deep Learning: A Comprehensive Survey* by Wang et al. [59] and the work of Li et al. [38] address the open challenges, opportunities, and state of research in deep learning on the edge in general.

Scoping to deep learning on the edge for Wildlife Classification purposes, in the previously mentioned 2022 study of Zualernan et al.[64], the authors provide an extensive literature study on advancements of CNNs and Wildlife Classification Research on the edge. In the remaining sections of their research, the authors propose an edge IoT architecture that uses deep learning to convey wildlife classification results to a mobile app. Their study includes wildlife data with 66400 images and trains the InceptionV3, MobileNetV2, ResNet18, EfficientNetB1, DenseNet121, and Xception neural network models. The authors found the Xception model to perform best, and when optimized, deployment on the Jetson Nano proved to be the most performant edge device. The authors acknowledge that unbalanced data remains a challenge, especially in the edge model domain, and suggest that techniques such as semi-supervised learning might be able to address this issue.

In the work of Cuhna et al. [19], the filtering of empty or "ghost" images on embedded systems is discussed. In their work they make a distinction between classifier and detection models, similar to the distinction between single-stage and multi-stage classification made previously, and conclude that the detection models are superior in their precision-recall performance, but are more costly to run on the edge. What is particularly interesting for our research is that we will use a detection model as a benchmark model to compare our novel model approach to, meaning that our research advances the ongoing discussion on the viability and trade-offs between detector models and classification models on the edge.

On the notion of detection models, Arshad et al. [8] present a wildlife identification and counting device which is able to track and count individual deer. Their method is based on the YOLOv3 object detection model and was deployed on an edge device. Without manual tagging or marking, the deer are counted by using discriminative correlation filters. Their approach aims to address the re-counting problem, a situation that occurs when the same animal is counted multiple times. Their results show that 17 out of 20 deer were correctly counted, taking a total of 0.3 computing hours compared to 18.5 manual hours of counting. This reaffirms the place of classification and counting of wildlife on the edge using deep learning as a factor to decrease the labor required for analysis by researchers.

In the study by Jia et al. [27], the authors use Neural Architecture Search based on Regression Trees (NASRT) to find CNN architectures for wildlife classification scenarios on the edge. Using NASRT, they are able to design the architecture specifically for resource-constrained devices and are able to adaptively change the architecture to fit the limited resources of specific edge devices. The results show to be able to compete with manually designed networks. While the research is relevant and interesting in the domain of finding and optimizing novel CNN architectures for animal classification on the edge, Neural Architecture Search is not a direction that will be ventured into in this research.

Outside of the camera trapping context, there exist an array of alternative approaches to Wildlife classification on the edge using embedded devices. Examples are the work of Gutierrez-Galan et al. on behavior recognition, classification, and monitoring using a collar-based device and a wireless sensor network[25]; and the work of Dominguez-Moreles et al. where the authors develop a hierarchical wireless sensor network using a collar-based device, motes(routers) and base-stations(coordinators) in order to classify and track the behavior of animals in the Doñana

National Park [21]. These studies show

2.2 Literature in the context of our objectives

In this section, we highlight studies that are especially relevant to our research objectives.

The 2022 study of Leorna et al., [37] evaluates the performance of the MegaDetectorv4.1 model using data from an ongoing camera trapping study in Arctic Alaska, USA. The authors give insights into the limitations of such models in terms of failure to detect wildlife based on animal sizing and distance. They describe a minimum detection size limit for camera motion detection sensors, the MegaDetector model, as well as humans. They found the minimum detection limit for a caribou in a camera trap image to be 600px for the camera sensor, around 60px for the MegaDetector, and 4px for human reviewers. For the purpose of this research, we can expect existing models to struggle with reduced input at the point where an input image is at a maximum of 60x60 in dimensions. These resolutions can be a useful starting point for the input dimensionality reduction section of this research, as they indicate the limitations of existing state-of-the-art models in terms of minimal detection limits.

Continuing the thought of input constraints in terms of image resolution, in the research of Koziarski et al., [34] the author evaluates to what extent low resolution impacts the performance of deep learning models for image recognition. Their results suggest that contemporary (i.e., traditional) neural network architectures are impacted significantly by reduced input image resolutions. They propose that super-resolution, that is increasing the resolution by artificial means, prior to classification can alleviate some of the impacts. However, this is only possible so long as the resolution of the image was not decreased too severely. For the context of our research, super-resolution is not relevant. Given this, the insight that contemporary neural architectures are impacted as such by decreased resolution further supports our goal of introducing a novel model approach capable of dealing with low-resolution images.

A different constraint to reducing the input dimensions is that of converting multi-channel to single-channel images, or RGB to gray-scale. In a study by Tabak et al., the authors use a model based on the ResNet-18 architecture to classify wildlife camera imagery [52]. Interestingly enough, the authors found that the model performance for images during the daytime performed almost just as well as the images taken at night (accuracy = 97% and 98%, respectively). This is promising as this indicates that possibly wildlife images can be analyzed and stored in gray-scale without much impact on performance, which can be beneficial in edge scenarios.

In the context of models for lightweight object detection and classification on the edge, Danish et al. [20] discuss challenges faced when analyzing video streams on the edge, specifically for object tracking. They find that existing sensors use high amounts of energy while continually tracking objects, and consequently propose a novel approach to track the movement of people while retaining a reasonable accuracy. Interestingly, they find that the Yolov5 family of models (n5 & n6 specifically), which is one of the models which will be employed in this research, work poorly on the edge as their latency can be comparable to other models, but their performance scores are poorer. Continuing with the thread of the use of the Yolo family of models in the context of our research, researchers Zhao et al. [63] perform a case study using the Yolo models in the context of real-time wildlife classification. Although the original Yolo architecture outperformed their architecture approach in terms of detection accuracy, by replacing the original backbone with a MobileNet network architecture, the researchers were able to increase the detected frames per second on a CPU-bound device significantly.

One of the questions addressed in this research is on alternative techniques to increase classific-

ation results on the edge. In a study by Okafor et al. [48], the authors compare the performance of deep CNNs and Bag of Visual Words (BOW) models for wildlife classification. The performance results on a novel dataset (Wild-Anim) showed that each of the CNN models (AlexNet and GoogleNet based) outperformed the BOW techniques significantly. Additionally, by reducing the breadth in the neural layers, the researchers are able to decrease the computing time of the CNN models while proving to be competitive with the unmodified architectures. In the BOW and modified HOG-BOW approaches, the RGB color space is used, and the authors suggest studying the effect of the use of different color spaces in combination with deep learning techniques. Their results give rise to the idea that alternative techniques to increase classification results on the edge likely will not replace deep learning approaches, but instead should be considered as complementary methods.

Chapter 3

Background

This chapter introduces a selection of techniques and models used in the methodology for answering our research questions. Some level of prior knowledge in the machine learning domain is assumed by the reader.

3.1 The Variational Autoencoder

3.1.1 Autoencoders

An Autoencoder is a feed-forward neural network that maps high-dimensional data to lower-dimensional data and reconstructs it back into higher-dimensional data. It consists of two parts: an encoder which encodes the higher dimensional data into the lower dimensional space (also referred to as the latent space), and a decoder which translates the data point from the lower dimensional space back into higher dimensional space. Thereby, it can act as a way of compressing an input into the latent space. The encoder and decoder are neural networks that learn a scheme to encode and reconstruct the input data. The learning optimizes this compression and decompression schema using a loss function based on the distance between the input and the reconstructed output. Since autoencoders are trained using a loss based on the reconstruction of the output, the model is unsupervised (i.e., the training data can be unlabeled). Example applications for autoencoders include data compression, feature extraction, and anomaly detection. Where the Autoencoder encodes the input deterministically, the VAE which is introduced in the following subsection generates a probabilistic representation of the data in the latent space.

3.1.2 Variational Autoencoder

The Variational Autoencoder is a type of autoencoder that learns a probabilistic representation of the input. The VAE was first introduced by D.P. Kingma and Max Welling in 2013 [33]. Figure 3.1 shows a high-level architecture of a VAE. The key feature of the VAE as compared to the Autoencoder which is relevant to this research is that the latent space is more disentangled and structured. This is due to the stochastic relationship between the input of the encoder and the input of the decoder. In addition to the latent space being more structured, the VAE allows for generating new samples of data using generated points in the latent space. For the purpose of this research, our focus lies on the structured relationship between input and location in the latent space. During training a VAE, we optimize for the ability to reconstruct the input image based on the input image in addition to the Kullback-Leiblker (KL) divergence, which helps ensure the latent space distribution.

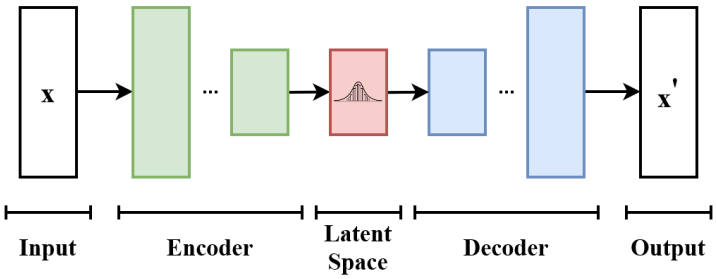


Figure 3.1: Basic VAE architecture. The input X is reconstructed as output X' by first compressing the input in the latent space. This is followed by sampling from the latent space and generating a reproduction from X as X'. Image: [55].

3.1.3 Latent space visualization

Visually representing the latent space by plotting data encoded by a VAE model can be useful to (visually) determine the extent to which encoded data is separable in the latent space. The main challenge in visualizing the latent space is the high dimensionality of the data in the encoded latent space. Although the encoder is a compression mechanism, the latent space may still be represented in dimensions that are not feasibly visualized without any prior processing (for instance a 128-dimensional vector). There exist multiple techniques for achieving this, most notably Principal Component Analysis (PCA) and t-SNE (t-Distributed Stochastic Neighbor Embedding).

This research makes use of t-SNE for reducing the number of dimensions of data that have been encoded to the latent space. t-SNE iteratively converts similarities between points to joint probabilities and uses KL divergence to minimize the distance between these joint distributions. For our purposes, we use this technique to reduce the data into 2-or 3-dimensional space. That way, the data can be represented using traditional visualization techniques. An example of a visualization based on T-SNE embeddings can be found in Figure 3.2.

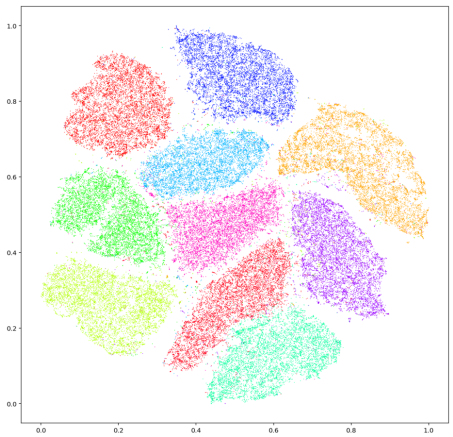


Figure 3.2: Example T-SNE embedding of the MNIST dataset, a dataset of handwritten digits. The digits are labeled by color, and a clear separation of the embeddings of the digits can be observed by the spaces between the clusters of digits. Image: [57].

3.2 Color and texture descriptors

Techniques that use color and texture descriptors for feature engineering could prove useful in the context of classification on the edge. We consider these techniques in the context of the third research sub-question on alternative supporting techniques for the classification of wildlife on the edge. In the work of Bianconi et al. [12], a historical overview and taxonomy of color and texture descriptors for visual recognition are given. In this study, the authors divide color and texture descriptors into data-driven (deep learning) approaches and theory-driven (hand-crafted) approaches. The theory-driven approaches are a-priori approaches where, regardless of the data, the operations used for feature engineering remain the same. In the context of our research, we theorize that if the a-priori-defined operations are relatively cheap in terms of computations, they could result in a cheap way to extract features from an image. This would assist in improving model performance on the edge while remaining *cheap* in terms of computational cost. Two a-priori techniques are of particular interest to us, which are the Color Histogram and Local Binary Patterns.

3.2.1 The Color Histogram

The color histogram is an a-priori technique for visual feature extraction. Although conceptually and technically simple, it has shown to be an effective tool to discriminate for color textures [12]. It captures the distribution of the colors in an image by counting the counts of the color values for each of the color channels in an image. In the 8-Bit RGB images used primarily in this research, this means the counts of each of the 256 values for each of the red, green, and blue color channels. Figure 3.3 shows an example of the extracted color histogram from the cat image.

Figure 3.3: An image of a cat with heterochromia, and the associate color histogram. The x-axis of the color histogram represent the RGB values, and the Y-axis the frequency. Image left: [58] Image right: [56].

3.2.2 Local Binary Patterns

Similarly to Color Histograms, Local Binary Patterns is an a-priori technique for extracting features from an image. In particular, it is a texture descriptor that works by generating a label (or binary code) for each pixel based on thresholding its NxN neighborhood, where $N = 3$ is what was first used by Ojala et al. [47] who introduced the technique in 1996. By using a sliding window approach, a binary code can be generated for each of the pixels in an image. Based on the binary codes (i.e., the local binary patterns), a histogram is computed which acts as our feature vector.

3.3 Models

In this section we give a brief introduction two of the models used in this research.

3.3.1 Yolo

The YOLOv5 (You Only Look Once) model is the fifth edition of an deep learning object detection model which is developed by Ultralytics [28]. It was released in June 2020 and showed state-of-the-art performance at the time of release [28]. It serves as the backbone for MegaDetectorv5, a well-performing public wildlife animal detection model [11]. In the context of single-stage and multi-stage models described in the Literature Chapter 2, the Yolo model is a single-stage model. Although both a bounding box and classifier model, it is a single-stage model as both the bounding box and classification of the object occur in one pass. The Yolov5 model series are in fact a collection of models ranging from 140M parameters in size down to 1.9M parameters in size. As Yolov5 has previously been used as the backbone for existing wildlife studies (e.g., [20]), and the MegaDetector model in particular, the Yolo models make for a useful baseline comparison model. The use of the Yolo family of models as part of our Methodology is described in Section 4.3.1 and the experimental setup can be found in Section 5.2.1.

3.3.2 GhostNet

In 2020, Han et al., [26] proposed a novel CNN module called the Ghost module. This is a 'plug-and-play' component which is able to upgrade existing CNNs by generating feature maps more efficiently by using cheap operations. For embedded or edge devices with limited computational resources, the increased computational efficiency can be especially beneficial. The authors show that a model using the ghostnet module can achieve better performance than MobileNetV3, a popular edge model, while having similar computational costs associated with the models. The application of the GhostNet in context of our methodology is discussed in Methodology Section 4.3.2 and the experimentation details are described in Section 5.2.2.

Chapter 4

Methodology

The methodology chapter covers the description of techniques used to answer our research and sub-questions. We begin by describing the relationship between the methodology and research questions. This is followed by explaining the data pipeline approach. Lastly, the classification models are addressed. In the next chapter, the experimental setup will be explained.

4.1 Methodology and research questions

This section covers a structural overview of the methodology and how it relates to the research objectives. We begin by taking a look at the complete overview in Figure 4.2. We start by explaining the model pipeline which gives a high-level view of the experiments. This is followed by the data pipeline in Section 4.2 (found below in the overview figure) and the classification models in Section 4.3, i.e., the remaining sections in the overview figure.

The top section in the complete overview is shown separately in Figure 4.1. This model pipeline is an abstraction of the high-level approach taken for each of the experiments of the research. Plainly put, each experiment involves classifying wildlife in camera trap images. Two types of input images are considered, that is, *Single camera trap source images* which are images that are not necessarily taken in sequence, and *Sequential camera trap source images* which are images taken in sequences of 3 or more images in a row. Each of the images in the experiments is run through a data pipeline, which processes the images according to various parameters. This data pipeline generates increasingly small images based on compression, resizing, and gray-scale conversion and is further described in Section 4.2. The next step in the model pipeline is the classification models.

Three classification models are considered. First, the Yolo models, which are purely used for evaluation purposes on the increasingly small input images; second the GhostNet models and their variants, used for the experiments with non-traditional approaches to supporting wildlife classification on the edge; and third, the VAE-SVM model which is our novel model approach. Each of the models classifies the images after which the results are saved. For each of the results, we focus on class-wise precision, recall, and F1 score as evaluation metrics.

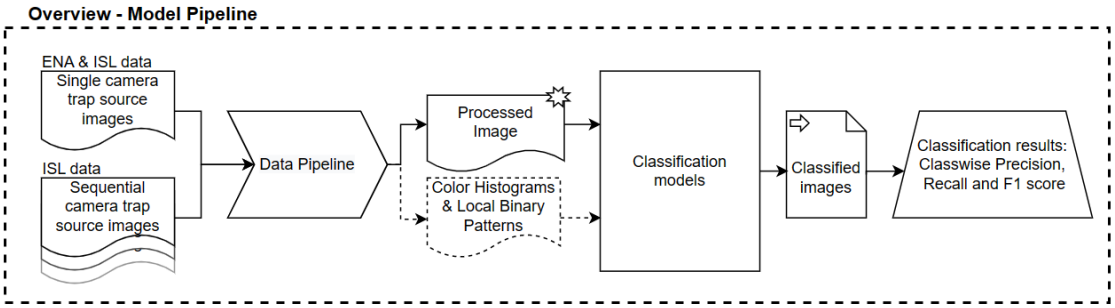


Figure 4.1: Model pipeline

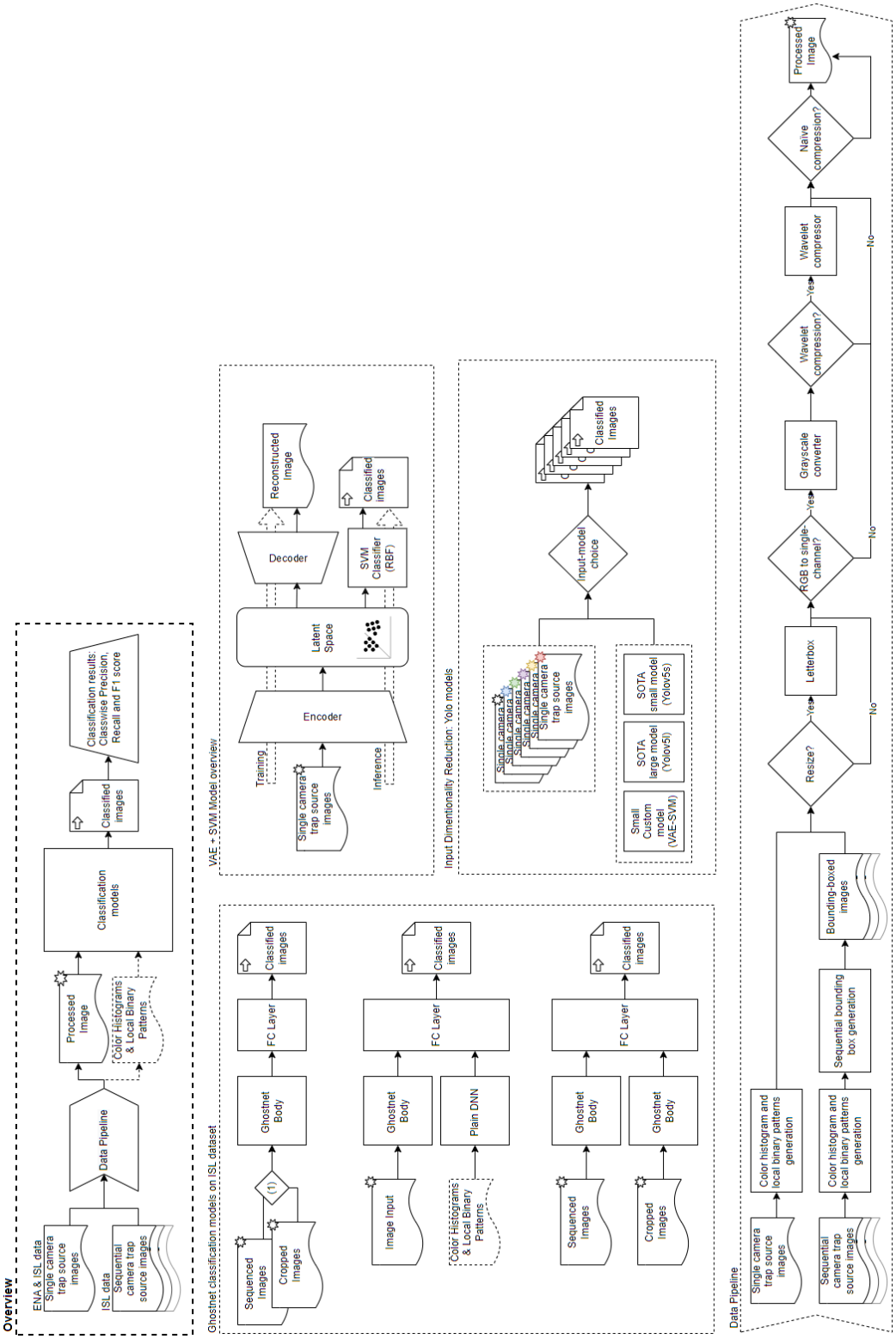


Figure 4.2: Complete overview of our experimental design

4.1.1 Approach to answering the research questions

The first research sub-question concerns reducing the dimensions of the input data. We have previously introduced the importance of decreasing input dimensions for deep learning networks in relation to the capability of running models on the edge. To briefly reiterate, reducing the input dimensions can be useful in particular in terms of reducing the computational cost of running the models, and alleviating computational and energy bottlenecks faced on the edge. The approach taken in this methodology is to structurally decrease the input dimensions of images by resizing, compressing, and gray-scale conversion using the data pipeline, as well as decreasing the number of total training samples. Using a variety of subsets with decreased input dimensions, we test the benchmark Yolov5s models to our custom models. This approach is followed up to the point where these models no longer function well given their input. Concretely put, this occurs when the classification accuracy is significantly decreased, meaning well below the original model capabilities, but meaningfully better performance than random guessing. In subsection 4.3.1 the approach to testing the influence of input dimensionality reduction on the Yolo models is described.

The second research sub-question aims to design a model capable of dealing with reduced input dimensionality. Traditional machine learning models are not designed to deal with very small, compressed, and gray-scale images by default, and we need to investigate their ability to cope with constrained input. Additionally, we design a model able to specifically cope with these constraints and aim to increase the classification capabilities of wildlife images as compared to current state-of-the-art models while being a small (i.e. edge) model. Concretely, the model should be able to deal with images of size 64x64 pixels in gray-scale. As the original images are up to 3000x2000 pixels in size, this is a significant reduction. The model which was designed for answering this research sub-question is the VAE-SVM model. This model is found in the overview in the overview Figure's subsection labeled *VAE + SVM model overview*. The design choices for this model are discussed in Section 4.4.

The third research sub-question looks at alternative techniques to improve classification on the edge. In particular, we look at background subtraction techniques to generate bounding boxes for wildlife, and extraction of local binary patterns and color histograms as features. In the Background Chapter 3.2, the a priori techniques of Color Histograms and Local Binary Patterns are highlighted. The background subtraction technique is used to create a bounding box around animals in a computationally efficient manner. Camera trap images taken in sequence can be used in combination with background subtraction techniques in order to create a bounding box around the animals. We investigate the effectiveness of these three techniques using the GhostNet model [26] and variations thereof. Experiments run with this model look at the effectiveness of applying these techniques in terms of their impact on the performance metrics described in the previous Section.

4.2 The data pipeline

We designed a data pipeline for a variety of purposes. Various subsets of the datasets are required to run the experiments, each with different specifications. Examples of such specifications are varying the number of color channels, adding compression, and reshaping the image. Therefore, a modular data pipeline is useful in creating these subsets. An overview of the data pipeline can be found in overview Figure 4.3. The image pipeline consists of two distinct parts, i.e., a feature extraction part, and steps to alter the images to fit dimensionality reduction purposes. The feature extraction part is used to extract local binary patterns and color histograms from the images. The dimensionality reduction part is used to compress, resize, and convert the images to gray-scale. The local binary patterns and color histogram features extracted are only used in the experiments with the GhostNet model. Images that are reduced in size using the dimensionality reduction

part of the pipeline are used throughout the experiments. Details on the created subsets and their naming can be found in the experimental setup Section 5.1.

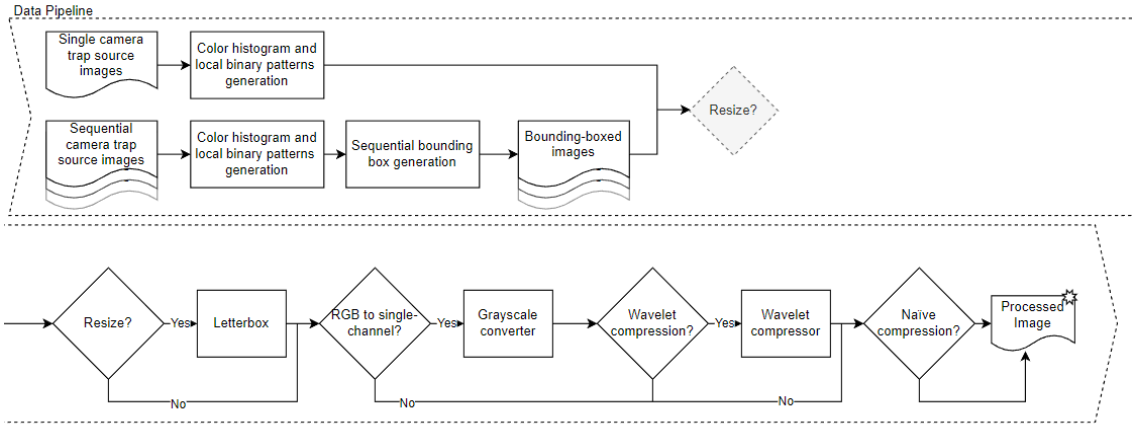


Figure 4.3: Data pipeline

4.2.1 Feature extraction

The input to the data pipeline is either a single camera trap image or a batch of images from the same sequence. The first step of the data pipeline is the creation of color histograms and local binary patterns from the unaltered source images, as at this point no information is lost yet due to resizing, compression, or other means. The color histograms and local binary pattern data are saved separately from the images. Considering the implementation, the three color channels are saved in 256 bins each, i.e, one bin for each of the 8-bit color depths. Local binary patterns are extracted with a radius of 1.5 pixels. The color histograms and local binary patterns are then appended into a single array and saved to disk for later use with the GhostNet experiments.

4.2.2 Background subtraction

One of the alternative techniques aimed at supporting wildlife classification on the edge is the background subtraction of wildlife images taken in sequence. For sequential images, this is the step following the feature extraction in the data pipeline. The main idea behind the background subtracted technique is that we can generate bounding boxes around animals in the wildlife imagery by using the subtracted background.

The background removal is implemented using the OpenCV library [14] using the MOG2 background subtraction algorithm and was implemented in Python 3.x. The algorithm to detect the backgrounds of the images can best be described by the pseudocode found in Algorithm 2. The input to the algorithm is a sequence of images, and the output is a sequence of those images where the background of each image in sequence has been removed. Using the first image in the sequence, the background of the second image is removed. This is repeated for images $3, ..., n$. Finally, using the last image, the background of the first image is removed. By taking the inverted background, i.e., pixels that are not part of the background, non-background object boundaries are highlighted. These boundaries are being used for generating the bounding boxes. The images will have some background noise at this point, which makes the generation of bounding boxes difficult. In order to remove the noise, a repetition of erosion and dilation was used. The number of erosion and dilation steps was experimented with, but in the implementation, two erosion and

dilation steps were chosen. An additional erosion and dilation step is performed during the bounding box generation. Varying the number of erosion and dilation steps has a significant influence on the effectiveness of creating the bounding boxes. As an example, balancing the number of steps determines whether or not small animals can be found (e.g. birds or rodents) or instead whether or not the resulting bounding box is larger than the animal due to background noise. This balance is shown in Figure 4.4. An example of both correctly and incorrectly cropped images due to the number of erosion and dilation steps is shown in Figure 4.5. Following the background subtraction step, the bounding box is extracted from the image by cropping around the highlighted object boundaries, and the image is cropped to the bounding box.

The bounding box generation is implemented by following the pseudocode of Algorithm 1. The algorithm takes a sequence of images as input and returns a sequence of images, where each image is cropped to the bounding box of the corresponding image in the input sequence. The backgrounds are generated with the algorithm described in the previous paragraph. Bounding boxes are generated by finding the contours of the object boundaries of the inverted background. The minimum and maximum x and y coordinates of the contours are used to generate the bounding box. An alternative technique would be to use the largest contour in the image and generate a minimum bounding box around it but based on the results of non-formal experimentation, that did not yield better bounding boxes. Generating contours is performed using the OpenCV library [14]. The implementation of the step *GetBox* is based on an algorithm posted by a user on a Stack Overflow discussion [30]. The implementation provides a set of contours given the subtracted background, which we then use to find the minimum and maximum contour coordinates as described above.

If the bounding box does not meet a minimum size threshold, it is disregarded. In the implementation, the threshold is set at 10 pixels both in the horizontal and vertical direction. Concretely, this means that if the minimum bounding box is smaller than 10 pixels in either direction, the image is not cropped. This is done to prevent cropping of images of only background noise, or otherwise unusable cropped images. If an image does not meet this requirement, the original image is returned. The last step before cropping is adding padding around the bounding box of 15 pixels on each of the sides while taking into account the boundaries of the source image. The algorithm concludes by returning the sequence of cropped images (referred to as 'sequenced' images).

A note on bounding box generation by means of background subtraction is that, in theory, this technique can be extended to all images captured on a single camera trap location by using images captured in nonconsecutive events. The camera trap location is often quite similar in terms of the background of the captured images. However, in practice, even slight alterations of the existing background (e.g. falling branches or changing lighting situations) make this unfeasible. Given that the environment can change significantly between images captured even on the same camera, the extracted background can no longer be a trustworthy indicator of animal movement. Consequently, only images captured concurrently in a sequence are considered in this approach.

4.2.3 Dimensionality reduction

The final steps in the data pipeline concern dimensionality reduction. As previously discussed, reducing the dimensions of an image, both in terms of file size and image size, can have a significant influence on the computational power required to classify the image. Therefore, reducing dimensions is a useful technique in edge classification scenarios. In our pipeline, we first resize the images, then optionally, the images are transformed into gray-scale or single color-channel images. Our implementation of wavelet compression only works on single-channel images, so the gray-scale images can be compressed with wavelet transformation in the step that follows. Lastly, the images can be compressed with regular or 'naive' compression.

Algorithm 1 Generating bounding boxes using backgrounds

```
1: procedure GenerateBoundingBoxes(seq)
2:     bgs ← GenerateBackgrounds(seq)
3:     output ← ∅
4:     for i ← 1 to length(seq) do
5:         image ← seq[i]
6:         bbox ← GetBox(bgs[i])
7:         if size(bbox) > (X_thresh, Y_thresh) then
8:             paddedImage ← addPadding(bbox)
9:             output.append(paddedImage)
10:        else
11:            bbox ← ∅
12:            output.append(seq[i])
13:        end if
14:    end for
15:    return output
16: end procedure
```

Algorithm 2 Generating backgrounds

```
1: procedure GenerateBackgrounds(seq)
2:     output ← ∅
3:     for i ← 2 to length(seq) do
4:         bgSubstractor ← MOG2(seq[i − 1])
5:         bg ← bgSubstractor(seq[i])
6:         bg ← ErodeDilate(ErodeDilate(bg))
7:         output.append(bg)
8:     end for
9:     bgSubstractor ← MOG2(seq[length(seq)])
10:    bg ← bgSubstractor(seq[1])
11:    bg ← ErodeDilate(ErodeDilate(bg))
12:    output.push(bg)
13:    return output
14: end procedure
```

Resizing

There are several methods of reducing the total number of pixels in an image, also known as resizing. For resizing in our data pipeline, we have chosen a letterbox resizing scheme (see Figure 4.6). The result of letterbox resizing is that the final image has the width and height of the target dimensions, while also retaining the aspect ratio. The drawback is that there are areas (or bars) on the top and bottom of the image for images that are empty. This is the result of resizing images with a larger width than height to square dimensions. All images in this research are *letterboxed*. A result of letterboxing images is that the resized images contain an area with no meaningful information, which is obviously a drawback. Other resizing options include cutting away the edges (commonly called 'cropping') which is more efficient in terms of information captured in the resulting image as we do not have bars at the top and bottom of the image, but can suffer from cutting away meaningful information such as wildlife at the edge of the images. Alternatively, having an irregular aspect ratio is an option, but this is not ideal as many models require square dimensions. Among these choices, the letterbox approach was deemed most appropriate as we do not want to lose information potentially by cutting it away, and the images need to be equal in width as well as height. The images used in the experiments are resized between a range of 640x640 down to 64x64. A note on the naming throughout this research, in the datasets Section 5.1, subsets which are labeled as cropped, are in fact resized using letterboxing. This is a misnomer that is consistently found throughout the experiments and their results.

An implementation detail on resizing and cropping is that one of the models used in the experiments, the Yolov5 model, requires a bounding box of the animal in addition to the classification label during training. The bounding box is a 4-tuple of the form (xmid, ymid, width, height). When resizing the images, the bounding box needs to be resized as well using the new ratio of the image.

Compression and conversion to gray-scale

The following step in the data pipeline is the option to convert the image from RGB to gray-scale (or to "single-channel"). Theoretically, converting an image to gray-scale would reduce the information stored in the image by a factor of three. While this is true in terms of information available to the consumer of the image (i.e. the model), conversion to gray-scale does not reduce the image size on disk by the same factor of three. This is due to how JPEG encoding efficiently stores RGB images. However, gray-scale conversion still leads to a meaningful reduction in image size while retaining relevant information for classification purposes. This is especially true in the wildlife setting, as many images are taken in lighting environments that do not benefit much from a full-color scale.

Following gray-scale conversion, the images are optionally compressed with wavelet transformation. Wavelet compression is a technique we chose to use in addition to regular or naive compression as it mostly aims at reducing noise components in the data. Consequently, we deem it a useful approach in reducing the size of images on disk while retaining useful information for classification. Both wavelet compression and regular compression are *lossy* compression techniques, meaning information is irreversibly lost after compressing.

In the final stage of the data pipeline, the implemented final step in the data pipeline is resizing the image to the desired dimensions using letterbox resizing and saving it to disk. During saving to disk, regular or "naive" compression can take place to reduce the storage required to store the images. *Naive* compression is implemented with the Python implementation of OpenCV [14]. OpenCV, in turn, makes use of LibJPG's *jpeg_set_quality()* which encodes the image using color subsampling, quantization of the DCT-coefficients, and Huffman-Coding, where the quality loss is controlled by the user.

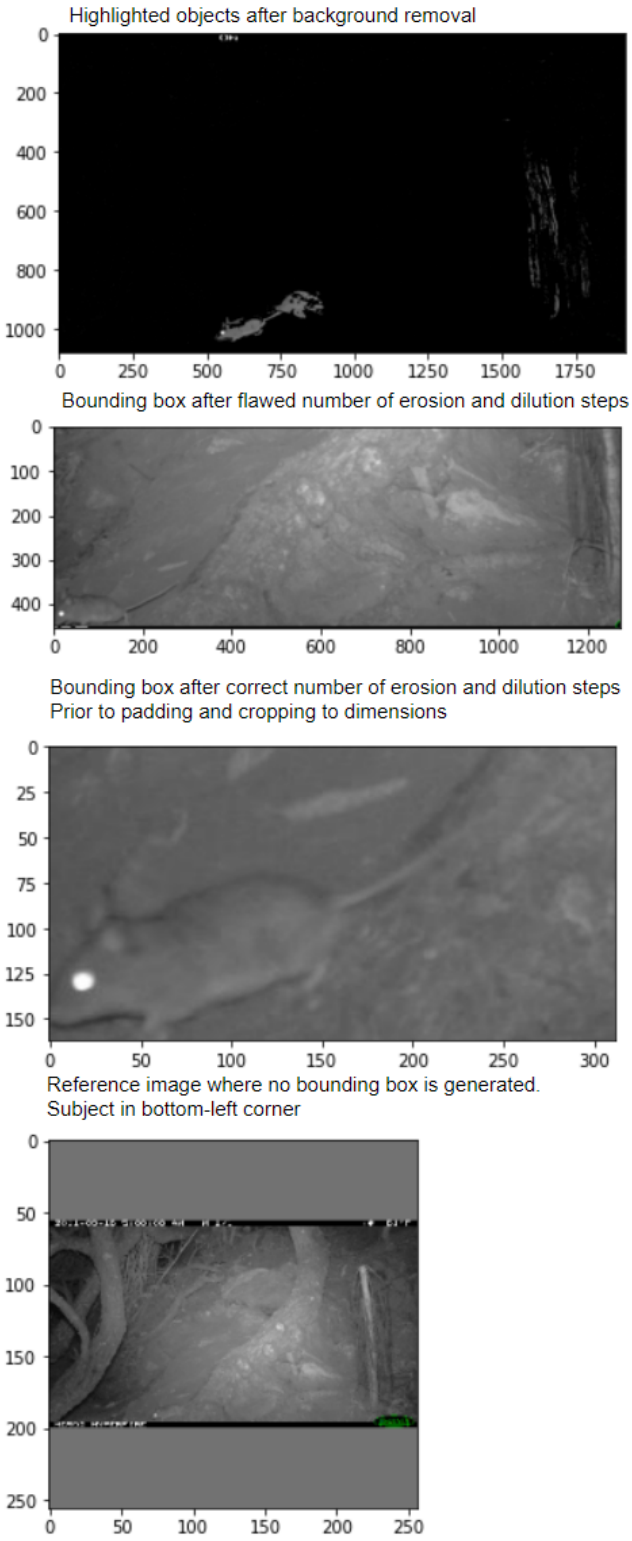


Figure 4.4: The influence of varying erosion and dilution steps in combination with noise in the background. The first (top) image shows the highlighted objects after removing the background. The second and third image show bounding boxes based on varying the amount of erosion and dilution steps, and the resulting generated bounding box (prior to reshaping to square dimensions). The final image is a reference image to showcase just cropping the image.

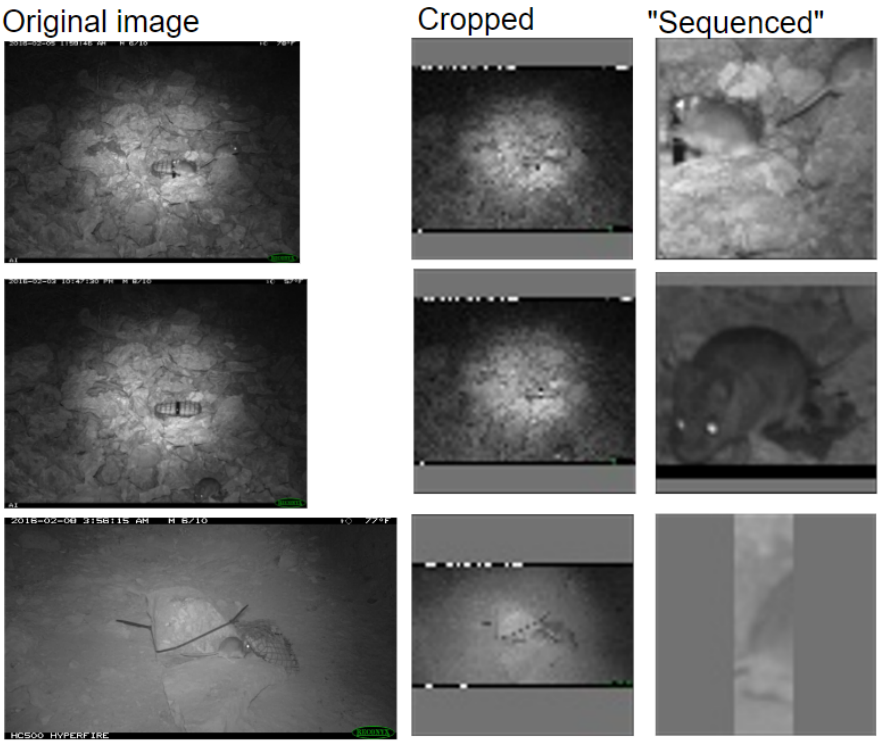


Figure 4.5: Example of the effect of using the "sequencing" trick on a selection of images with rodents. The first two sequenced images correctly crop the image around the rodent in the image, but the third example does not correctly crop around the rodents.

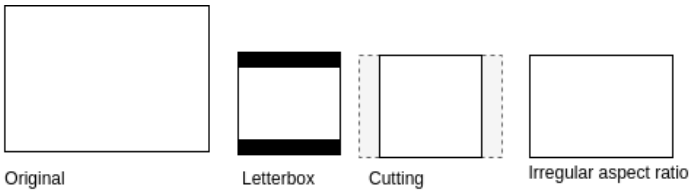


Figure 4.6: A selection of image resizing techniques. In this research, we make use of the letterboxing scheme, which introduces borders to fill the blank space while retaining the aspect ratio of the original image within the borders.

4.3 Classification Models

4.3.1 Yolo models for evaluating dimensionality reduction

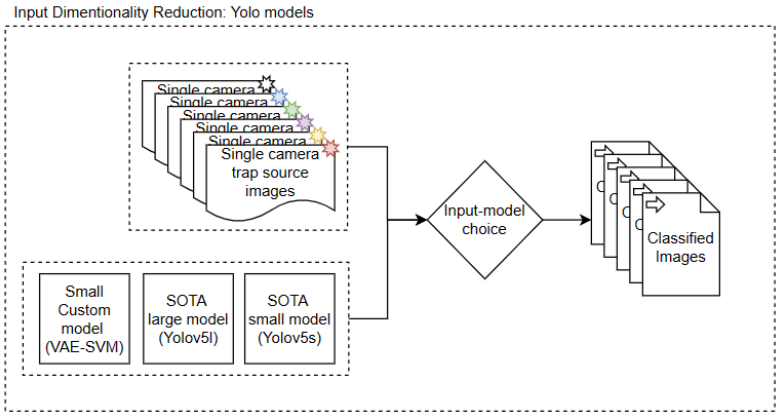


Figure 4.7: Overview of the input dimensionality reduction experiments

Reducing the input dimensionality can drastically reduce the computational load of image classification using neural networks. In the previous section, we discussed the various techniques which can be used to reduce the input dimensionality. Using varying parameters for the steps in the data pipeline, we construct a set of subsets with varying image sizes, compression amounts, and color channels (RGB and gray-scale). The constructed subsets can be found in Section 5.1. The object detection and classification model Yolov5 [28] has been trained and evaluated on these datasets. Two Yolov5 models have been used in this research; the large model (Yolov5l) and the smaller model (Yolov5s) fit for edge computing scenarios. The smaller model was considered since it could be run on the edge, and the larger model was used as a benchmark for comparison purposes. There exists an even smaller model, Yolov5n, but the size of Yolov5s to the GhostNet models is more comparable, so Yolov5s was chosen. Details on the experimental with Yolov5 can be found in Subsection 5.2.1.

The reasoning behind choosing the Yolov5 large and small models as a benchmark or "golden standard" is in part due to the high performance and reliability. The Yolo models have been widely used and studied, and at the time of release, Yolov5s received state-of-the-art results on benchmark datasets. In addition, Yolov5's use as the backbone of MegaDetector [11] not only showed its success in the field of wildlife camera trapping but also resulted in a large amount of data available for training. This is true in particular for object detection models which require bounding box information in addition to labels. The available data in combination with the straightforward implementation made it an accessible choice for this research. These factors made us choose Yolov5 as the benchmark for the input dimensionality reduction experiments.

4.3.2 GhostNet model and its variants

Classification of wildlife imagery using deep learning models can be costly in terms of computational power. In edge scenarios, resources in terms of computational power and energy are sparse, so the cost of classification should be decreased to make the classification of wildlife on the edge a viable approach. Traditional (Non-deep learning) techniques can potentially assist existing deep learning models to improve their classification capabilities without adding much overhead in terms of computational cost. In this section, we propose the use of two existing techniques of feature extraction, namely local binary patterns and color histograms, in addition to the use of the back-

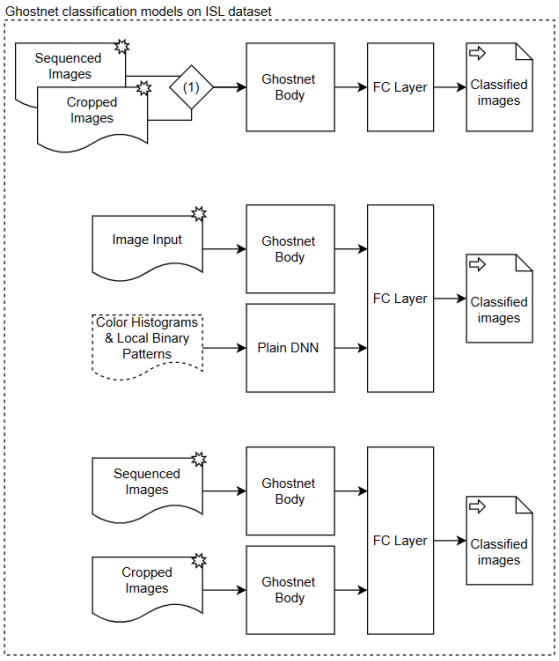


Figure 4.8: Overview of experiments on alternative techniques for improving wildlife classification on the edge

ground subtraction and bounding box generation mentioned in earlier sections in the Methodology, as the alternative techniques to support wildlife classification on the edge. The model used in the experiments to support this purpose is the "GhostNet" model [26]. In the overview in Figure 4.8, the four experiments run with the GhostNet model are shown.

The reason for choosing the GhostNet model is that we needed a model to adhere to the following requirements. The chosen model body and head should be easily separable; that is, the model body will be used to extract features from the images, but we want to be able to add an additional feature extractor to the model. The features extracted both by the original feature extract and the additional one will be concatenated before passing through the fully connected layer (i.e., the model head) which acts as the classifier for the image. An additional requirement is that the model needs to be sufficiently small in terms of the number of parameters to be considered an edge model. It should be possible to meaningfully compare to the Yolov5s model in terms of model size and computational efficiency. The GhostNet model meets each of the requirements, and consequently, is a logical choice for our purposes.

For running the experiments, we propose three different setups using the GhostNet model. First, the unaltered GhostNet model; the implementation was taken from the Efficient-AI-Backbones repository [36]. This unaltered version is used as a baseline for the GhostNet model and will be used to compare the performance of training and classifying GhostNet on the regularly cropped/resized images as well as the sequenced images. In the overview figure (Fig. 4.8) this is the first of three model setups.

The second version of the GhostNet model adds a deep neural network to extract features from the color histograms and local binary patterns. This neural network is trained alongside the original body of GhostNet, as it is added to the internal architecture of GhostNet. The neural network for extracting features from the color histograms and local binary patterns is a relatively plain deep neural network. It has two main components, which consist of a series of two fully

connected (FC) layers, two dropout layers to reduce overfitting during training, and two ReLU activation functions. The first fully connected layer takes an input size of 804 and results in an output of 120. The second FC layer takes an input tensor of size 120 and produces a dynamic output size. The output from both the GhostNet feature extractor, as well as the Color Histogram and local binary pattern feature extractor, are concatenated before they are passed to the fully connected layer. The size of the final fully connected layer in GhostNet is adjusted to take into account the concatenated results from the feature extractors.

The third version of the GhostNet model duplicates the GhostNet body in order to classify two images at once. The two images used for input are the 'sequenced' images which are cropped to the bounding box of the animal based on background removal, and the regularly resized ('cropped') image. The intent of this model is to be able to use both the information available in the sequenced as well as the cropped image. Similarly to the second GhostNet model, the final fully connected layer is adjusted to be able to process the extracted features from both images.

The first version of the model is run twice, once with the cropped subset and once with the sequenced subset. The second version is run once with the cropped subset and with local binary patterns and color histograms of the images. The final version is run with both the sequenced and the cropped images.

4.4 A novel approach: VAE-SVM model

In this section, we propose a method for the classification of wildlife animals in low-dimensional imagery using a combination of a variational autoencoder (VAE) and a support vector machine (SVM). The VAE is trained using a loss function based on the reconstruction loss and KL-Divergence and the training images, and the SVM is trained on the latent space representation of the images in addition to the labels of the images. This means that the training of the VAE is unsupervised, but the training of the classifier (the SVM) is not. By using the latent space representation of images processed by the encoder model, we compress features from the images in N-dimensional space. Using the distance between points (i.e. between the encoded images) in the latent space, the SVM then assigns them to a class. To better understand and visualize the learned latent space, we use the t-SNE algorithm. This is a dimensionality reduction technique that can plot the high-dimensional latent space onto a 2-dimensional or 3-dimensional plane. By labeling the classes with distinct colors, we are able to distinguish between the classes and are able to visualize the relationship between the different classes of animals in the learned latent space of the VAE.

Given the low resolution of the images used in the model (64x64) compared to original source images (up to approximately 3000x2000), certain classes will be difficult to classify given the limited information available in the 64x64 image. To address this issue, the aim is to let the VAE model learn the properties of the camera trap locations so that it can effectively distinguish background from wildlife. The VAE model trained on the images learns to reconstruct the images by encoding the images to a latent space and decoding the latent space representation back into an image. Thereby, it learns to effectively compress the information present in the image in the latent space representation, as the decoder model is only able to use the information present in the latent space representation to reconstruct said image.

After the unsupervised training of the VAE model on the training images, the model is frozen and saved. The images are then passed through only the encoder model and the resulting latent space representations are saved. These representations are analyzed with t-SNE and plotted with the labels. Next, an SVM classifier is fit on the latent representations of the images along with the labels. Finally, for each class, the class-wise accuracy, precision, recall, and F1 scores

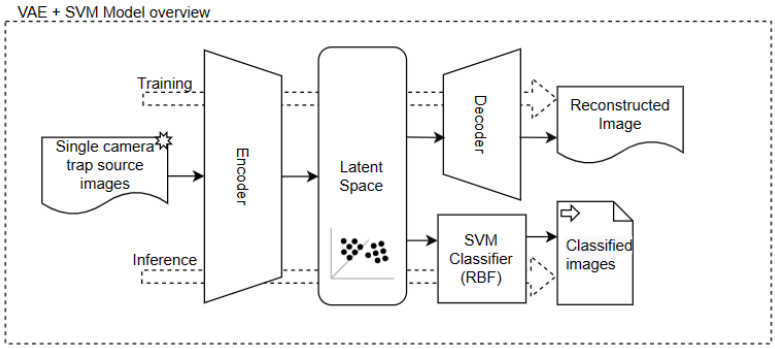


Figure 4.9: Overview of experiments using VAE-SVM

are calculated and recorded. For the SVM kernel, we have chosen to use an RBF kernel for the classification, which is a widely used kernel shown to perform well in a variety of situations. Other kernels may also show to be effective, and while informal experiments indicated slight performance differences in varying the kernel, it was out of scope to optimize for the specific kernel. As with the other models, we have avoided optimizing the models by means of hyper-parameter turning or extensively experimenting with different model architectures of training setups in order to find the model which performs best in our scenario.

Chapter 5

Experimental Setup

In the previous chapter, we described our research methodology, laying the foundation for our experiments and approach. In this chapter, we dive deeper into the setup of the experiments. We start by describing the selected datasets in Section 5.1, followed by describing the models to be used in our experiments in Section 5.2.

5.1 Datasets

In this section, we describe the ENA24 [61] and Channel Islands datasets [18], which are the datasets used in the experiments. We start by explaining the choice of datasets, followed by the data cleansing and subset creation using the data pipeline.

In order to select a dataset that fits the needs of the research, a set of criteria was established based on requirements from the problem statement as well as requirements are given the research methodology. The following criteria are considered:

- *Representability:* The selected dataset should represent datasets that would be encountered by wildlife researchers in real camera trapping (edge) scenarios, i.e., no significant preprocessing has taken place.
- *Reliability:* The labels and annotations should be reliable and consistent.
- *Availability:* The dataset should be public and available to the general public and have previously been subject to other research for a fair comparison.
- *Label format:* The labels should contain information about the class of the image (i.e. not just 'empty' or 'nonempty'), and in addition, contain information about the bounding box of the animal(s) in the image. Bounding box information is a requirement for the Yolo models. Only one bounding box per image is required even though there might be multiple animals present.
- *Size:* The dataset should be sufficiently large in size to allow for training the models.

We chose to use datasets from the LILA BC (Labeled Information Library of Alexandria: Biology and Conservation) data repository. This repository contains datasets related to biology and conservation and has a proven track record in terms of research. For instance, a selection of datasets from this repository has previously been used to train the MegaDetector model series [11]. Many of the datasets found in this repository adhere to the criteria set previously, as they are public and accessible, reliable, and consistently labeled using a proprietary "Coco Camera Traps" format [11] (described below).

5.1.1 ENA24 dataset

The ENA24 dataset [61] is a dataset containing a selection of 23 classes from Eastern North America. The dataset contains 8789 publicly available images with annotations following the COCO Camera Trap format [11], which is a metadata format very similar to that of the popular Common Objects in Context dataset [39]. The COCO camera trap format contains metadata of the image in JSON format including information about the sequence an image may belong to, the bounding box in the image, the class label of an image, and possible annotations. The publicly available ENA24 dataset contains no images of humans as they were removed for privacy reasons.

During the data cleansing, the human class labels were removed and the gap in the labels was replaced with the label of the class with the highest index, as a gap in labels will cause issues with the Yolo model training. No corrupted images or labels were found during data cleansing. Following the previous steps, the data was split into subsets of varying sizes. These splits were made using a stratified split on the classes, meaning the original distribution of classes was kept. First, 20% of the training images were split into subsets as the test subset, this subsets is never seen during training. Next, from the remaining images, 10% was set aside for validation purposes. Finally, the remaining images were used for the training subsets. These subsets include 100, 50, 20, 10, and 5 percent of the total training images. Figure B.1 in the appendix shows the stratified splits for each of the subsets and their classes. From an implementation perspective, the locations of the images and the labels of each of the subsets were saved in the *Pickle* Python data format. These *Pickle* files were then used in the data pipeline to generate the actual subsets based on a variety of input dimensionality reduction parameters. As the set of images used for the validation and test splits are defined at this step, all the transformations and processing steps that take place in the data pipeline are also performed on the validation and test sets. In the rest of this chapter, the created training subsets are described, but the validation and test sets undergo the same pipeline transformations.

For the various experiments, we require datasets that undergo a variety of transformations to change their input dimensionality, as well as to extract features such as local binary patterns and color histograms. The final created training subsets for the ENA dataset are shown in Table 5.1. The subsets were created using the parameters shown in the table header. The naming of the subsets indicates the parameters chosen in the data pipeline as well as the training set sizes.

To distinguish between the set of subsets, vertical lines were added in Table 5.1. The first set of subsets decreases the number of training images without changing the image dimensions or compressing. They range from ENA640xCropRGBTrain100 to ENA640xCropRGBTrain5 and contain 100-5% of the original set of training images. The second set of subsets introduces compression and grayscale conversion. We first turn the images gray-scale, followed by grayscale with wavelet compression, grayscale with naive compression, and finally grayscale, wavelet, and naive compression. Lastly, the final set of subsets decreases the image resolutions. For the ENA24 dataset, the image dimensions were decreased from 640x640 down to 64x64.

The full original training dataset was 4.8 GB in size but was not actually used for training the models. This is because the images in the original were larger than 640x640, and we decided 640x640 to be the maximum image size for the experiments. Therefore, in the results, the original training set is not shown and instead the results are compared between the ENA640xCropGNTrain100 and the other subsets. The reason for including ENA224xCropGWNTrain20 (as opposed to Train100) was to compare the limits of image compression and dimensionality reduction on model performance in comparison to the full training set (ENA64xCropGNTrain100) with smaller dimensions while having a similar final subset size (10.6MB and 8.0MB respectively). For the smallest images of 64x64 pixels, wavelet compression was omitted as wavelet compression introduces a black border as an artifact at the top and left sides of the images. As the images get smaller, the proportion between the compressed image and the artifact increases, and for 64x64

images the file size reduction was deemed too little in comparison to the compression artifacts. A table showcasing the validation and test subsets can be found in the appendix in Table B.2.

Table 5.1: ENA24 subsets

Subset name	Subset size	Image dimensions	Images	Size per image (KB)	Channels	Compression
ENAORIGxRGBTrain100	4.8 GB*	Original	6154	779.98	RGB (3)	None
ENA640xCropRGBTrain100	909.3 MB	640x640	6154	147.76	RGB (3)	None
ENA640xCropRGBTrain50	455.2 MB	640x640	3077	147.94	RGB (3)	None
ENA640xCropRGBTrain20	181.6 MB	640x640	1230	147.64	RGB (3)	None
ENA640xCropRGBTrain10	90.4 MB	640x640	615	146.99	RGB (3)	None
ENA640xCropRGBTrain5	44.6 MB	640x640	307	145.28	RGB (3)	None
ENA640xCropGTrain100	832.3 MB	640x640	6154	135.25	Grayscale (1)	None
ENA640xCropGWTrain100	735.9 MB	640x640	6154	119.58	Grayscale (1)	Wavelet Transform
ENA640xCropGNTrain100	362.4 MB	640x640	6154	58.89	Grayscale (1)	Naive compression
ENA640xCropGWNTrain100	314.6 MB	640x640	6154	51.12	Grayscale (1)	Wavelet & Naive
ENA224xCropGWNTrain100	53.4 MB	224x244	6154	8.68	Grayscale (1)	Wavelet & Naive
ENA224xCropGWNTrain20	10.6 MB	224x244	1230	8.62	Grayscale (1)	Wavelet & Naive
ENA64xCropGNTrain100	8.0 MB	64x64	6154	1.30	Grayscale (1)	Naive compression

5.1.2 Channel Islands dataset

The full Channel Islands dataset [18] is roughly 87 GB on disk and contains roughly 240k images with 6 classes, including an empty class. The images are taken from 73 camera trap locations in the Channel Islands in California. Similar to the ENA24 dataset, images containing humans are not publicly made available for privacy reasons. As the full dataset is quite large, only smaller subsets were used for the experiments. For each of the experiments, only 5% of the data was used for training. During the splitting of the data, images were set aside for validation and testing. As the Channel Islands dataset is used in combination with the background subtraction technique from Section 3, we were only interested in images taken in a sequence. Consequently, in the data cleaning section, the metadata was filtered to only contain images that are part of a sequence of three or more images. This means we disregarded all images which were not part of such a sequence.

In a similar fashion to the ENA24 dataset, subsets were created using a class-wise stratified split, keeping the class distributions roughly the same. Since we are using a sequence of images, stratification is a non-trivial task. For the channel islands dataset, class-wise stratification was performed by assigning a class label to each collection of images in a sequence, that is, each sequence of images was assigned one label. The most common class was chosen, where the non-empty class was ignored in the counts to prevent mislabeling sequences as fully empty. In case only empty class labels are found, the collection of images was assigned to the empty class. Using this set of sequences, a class-wise stratification can be performed. Since the dataset only contains a few images with multiple classes throughout a single sequence (non-formal data exploration), this stratification technique ensured the original class distribution is closely retained.

In Figure B.2 the stratified subset splits are shown. From these figures, it is clear that the dataset is heavily skewed, Certain classes are much more common than others, this imbalance is interesting to our research as it represents a realistic distribution of classes that could be found in real-world applications. Additionally, since the dataset contains fewer classes than ENA24, and also contains images taken in sequence, it makes the Channel Islands dataset a valuable dataset for testing our custom models in varying contexts.

The channel islands training subsets are shown in Table 5.2. In a similar fashion to the ENA subsets table, the header of the table indicates the parameters used in the dataset creation pipeline. The channel islands datasets were primarily used in order to evaluate the influence of alternative techniques to assist wildlife image classification such as the use of Local Binary Patterns, Color Historagrams, and background subtraction. Therefore, the images in the subset were not compressed or converted to grayscale, as the alterations are part of the ENA experiments. Only the stratified "Train5" subsets are used for training as described in the previous Section (5.1.2). The

validation and test subsets can be found in the appendix in Table B.1.

Table 5.2: Channel Island subsets

Subset name	Subset size	Image dimensions	Images	Avg. size (kb)
ISL224xCropRGBTrain5	143.8 MB	224x224	7126	20.180
ISL224xSeqRGBTrain5	129.4 MB	224x224	7126	18.159
ISL64xCropRGBTrain5	18.9 MB	64x64	7126	2.652
ISL64xSeqRGBTrain5	17.9 MB	64x64	7126	2.512

5.2 Models

5.2.1 Yolov5

The Yolov5 model is introduced in Background Section 3.3.1 and its use in the Methodology are introduced in Section 4.3.1. In the *Input Dimensionality Reduction* section of Figure 4.2 shows a basic diagram of the input dimensionality reduction experiments. First, we compare the performance of Yolov5s and Yolov5l [28] on a mostly unaltered dataset. The only change from the original dataset and the first dataset is that the images are resized to 640x640 which is the highest resolution of images considered in the research. The Yolo models are trained on the subsets with pre-trained weights as opposed to being trained from scratch, as this is suggested for small to medium-sized datasets as ours by the authors. The experiments, which were run using Yolo followed the subsets shown in Table 5.1 and Table 5.2. The initial experiments focus on reducing the number of training images, followed by the effects of gray-scale conversion and compression, and finally a combination of each of the reduction techniques. For the results, we only consider the classification results; i.e., class-wise precision, recall, and F1 score.

Each Yolov5 experiment was run with a batch size of 16 running for 100 epochs. The experiments were run by invoking the *train.py* script in the official Yolov5 repository from Ultralytics on GitHub. The model with pre-trained weights from training on the COCO [39] was used. The training was followed by evaluating the performance on the test sets. The model on the main branch with commit hash "7d46c6923592861aa123802c8860dfafc15d0fe1" was used for the experiments. The experiments were run on a remote cluster with CUDA/11.3.1 on a shared GPU. For the smallest of the data subsets, performance was also evaluated on a local machine to compare inference speed with the VAE-SVM models on the same hardware.

5.2.2 GhostNet and its variants

The GhostNet [26] model experiments and its variants are shown in Figure 4.8. The GhostNet model architecture was taken from the *Efficient-AI-Backbones* GitHub repository owned by *Huawei-Noah* [1]. The experiments with single input images use the *default* GhostNet (PyTorch) architecture taken from the repository. The experiments with "Dual GhostNet" altered the GhostNet by taking images as their input, each image runs through its own model body. Before passing to the fully connected layer which is adapted for the new tensor length, the tensors are concatenated. The GhostNet-Hist-LBP model takes a similar approach but instead concatenates the single image tensor and the resulting tensor from passing the color histogram and local binary patterns through a DNN.

From trial runs, we observed that GhostNet converged rather quickly within 1-3 epochs. Each of the final experiments was run for 10 epochs, where the models should have converged while no significant overfitting was taking place. The PyTorch built-in Adam optimizer was used with a learning rate of 0.02, with a batch size of 16. The GhostNet experiments were also run on the

remote cluster with CUDA/11.3.1 on a shared GPU.

5.2.3 VAE-SVM model

The diagram in Figure 4.9 from the Methodology (Chapter 4) shows the VAE-SVM model and experimental architecture. Experiments run with the VAE-SVM consist of two training phases, first to train the VAE model, and next to train the SVM classifier. For the VAE model, a "plain" VAE architecture was used. The implementation for the default VAE found on the PyTorch-VAE GitHub repository [4] was used. For the implementation of the SVM, the default implementation from SciKit-learn [50] was used with the built-in RBF kernel.

The VAE model is trained on the remote cluster with CUDA/11.3.1 on a shared GPU. Since no labels were used at this step, this is unsupervised training. The VAE model was trained for 200 epochs. No hyperparameter tuning was performed aside from a small number of initial trial runs for which the purpose was to obtain a working model. The chosen learning rate is 0.005 without any weight decay, and the Kld weight is set to 0.00025. The SVM model was fit on a local device. After training the VAE model to convergence, images were passed through the encoder and stored. Using the image encodings and their labels, the SVM was trained on the encoded data. The default parameters were used for the SVM kernel, meaning $C = 1$ and $\gamma = 1/(n\_features * X.var())$.

5.2.4 Comparison of experimental results

Before we address the results in the Chapter which follows, it is important to consider that the Yolov5 model is a classifier model which also detects object boundaries, and the other models used are purely classification models. As an object detection model, Yolov5 is designed to locate the objects in an image, as well as classify those objects, whereas the classification models are only trained to classify the object found in the image. In particular, when multiple objects are present in an image, comparing the performance of the classification model and object detection model is not as straightforward. The object detection model may detect and classify the wrong animal when two different animals are present, or the classification model might be able to predict accurately. However, images with two species are uncommon in the datasets used. For images that are very small in dimensions (e.g. 64x64), detecting objects in these images can be difficult- especially when the objects are small. This should be considered during the comparison of the Yolo models and the other models. That being said, addressing the shortcoming of existing models such as Yolov5 is precisely one of the aims of the experiments in this research. Results are compared using (average and weighed) F1 scores, precision, and recall regardless of the model used. The accuracy of the generated bounding box is not taken into consideration for the results.

Chapter 6

Results

In this chapter, we present the findings of our experiments aiming to improve the viability of the classification of wildlife on the edge. In the previous chapters, we have split the experiments into experiments that look at the influence of dimensionality reduction of the wildlife imagery on existing models, followed by alternative techniques which can support the classification of wildlife on the edge, and finally, a model which is able to deal with the constrained input images. This chapter will follow the same structure for presenting the results. Concretely, this means the following: first, the results of the experiments with the benchmark model yolov5 are presented. These results showcase the influence of input dimensionality reduction on the performance of the model. Next, we present our findings with the GhostNet model and its variants, experimenting with alternative methods to improve wildlife image classification on the edge. This includes background removal and bounding box generation using a sequence of images and using local binary patterns and color histograms as additional features. Finally, the performance of the VAE-SVM models is analyzed, which is the lightweight model able to cope better with the highly constrained input images. All of the results shown are on the test sets of the respective datasets. For all results on a class-wise level for both datasets, please refer to the appendix for the results, found in Appendix A.

6.1 Input Dimensionality Reduction

Table 6.1: Model sizes in terms of trainable parameters

Model	Trainable parameters
Yolov5l	46.5M
Yolov5s	7.2M
GhostNet	3.9M
GhostNet-Hist-LBP	4.0M
Dual GhostNet	7.8M
VAE-SVM-4M	4M*
VAE-SVM-665k	665K*

*Combined encoder & decoder weights.

In this section, we present the results of the input dimensionality reduction experiments. Table 6.1 shows the model sizes in terms of trainable parameters. The Yolov5s, GhostNet model, and VAE-SVM-4M model are quite similar in terms of the number of trainable parameters, and consequently, it is fair to compare their performance in an edge setting. The trainable parameters for the VAE models include the parameters for the SVM encoder and decoder. After training, inference can be performed using just the encoder which results in 50% fewer parameters, or 2M parameters and 344k parameters for the VAE-SVM-4M and VAE-SVM-665k models respectively. The VAE-SVM model parameters do not include the weights (coefficients) of the SVM, which are used to construct the decision boundary. The SVM weights account for $\leq$ 5k parameters in

practice.

Table 6.2: Summary of experimental results for input dimensionality reduction with ENA24 training data.

Model	Subset name	Subset size	Precision	Recall	F1
Yolov5s	ENA640xCropRGBTrain100	909.3 MB	90.24%	92.64%	0.913
Yolov5s	ENA640xCropRGBTrain50	455.2 MB	86.96%	89.10%	0.879
Yolov5s	ENA640xCropRGBTrain20	181.6 MB	84.85%	79.61%	0.819
Yolov5s	ENA640xCropRGBTrain10	90.4 MB	75.55%	70.12%	0.718
Yolov5s	ENA640xCropRGBTrain5	44.6 MB	66.67%	60.72%	0.629
Yolov5s	ENA640xCropGTrain100	832.3 MB	90.55%	91.97%	0.912
Yolov5s	ENA640xCropGWTrain100	735.9 MB	89.81%	92.97%	0.912
Yolov5s	ENA640xCropGNTrain100	362.4 MB	90.90%	91.31%	0.910
Yolov5s	ENA640xCropGWNTrain100	314.6 MB	89.51%	93.03%	0.911
Yolov5s	ENA224xCropGWNTrain100	53.4 MB	89.99%	86.48%	0.881
Yolov5s	ENA224xCropGWNTrain20	10.6 MB	78.69%	71.36%	0.745
Yolov5s	ENA64xCropGNTrain100	8.0 MB	65.06%	49.94%	0.540
VAE-SVM-665k	ENA64xCropGNTrain100	8.0 MB	67.38%	63.69%	0.634

Table 6.2 is a collection of the weighed precision, recall, and F1 performances of the experiments run on the ENA24 dataset [61] relating to input dimensionality reduction. In this table, we did not include the comparison between the large and small Yolov5 models. This comparison between the Yolov5s and Yolov5l model can be found in the Appendix Table A.1. In section 5.1 we introduced the created ENA subsets while explaining the data pipeline steps and their naming (Table 5.1). Each row represents an experiment run on one of the ENA24 subsets. The full results which include the class-wise scores are shown in the Appendix Tables A.2 and A.3 for the first five rows, rows four through eight are found in Appendix Tables A.4 and A.5, and rows nine through eleven can be found in Appendix Table A.6. The final row shows our novel VAE-SVM model approach and its performance on the smallest created subset, of which the class-wise results are found in Appendix Table A.7.

6.1.1 Reducing the number of training images

The first five rows in Table 6.2 show the results of reducing the number of available training images in the training set. A visual representation of experiments is shown in Figure 6.1. The total number of training images was reduced from 6154 to 307 total images. It is not surprising that decreasing the number of training images will decrease the weighted precision, recall, and F1 scores. This decrease in performance is expected as the model is trained on fewer samples. Classes with fewer initial samples are affected more by the reduced number of training images, which is an expected result. We observe this in the heatmap Figure 6.1 by taking a look at the support of a class (indicated by the number in the parentheses) and the decrease in F1 score with fewer input images. In this Figure, we see that certain classes were more affected by the reduction than others. For instance, the F1 score of the *Bobcat* class was reduced to *0.28* from *0.92*, while the *Eastern Chipmunk* class had fewer training samples but was only reduced to an F1 score of *0.54* from *0.93*. To emphasize the most important (and expected) finding, fewer training images results in reduced model performance, especially for classes with lower support in the training data.

6.1.2 Effects of gray-scale and compression

In addition to reducing the number of training images, we also experimented with compression and gray-scale conversion. Rows four through eight (i.e., the second division) in Table 6.2 corresponds with what is shown in Figure 6.2. They show the effect of reducing the input dimensions by means of compression and gray-scale conversion. Taking a closer look at the subset sizes when saved on disk as seen from the Experimental Setup Table 5.1, an almost 3 times reduction can be achieved

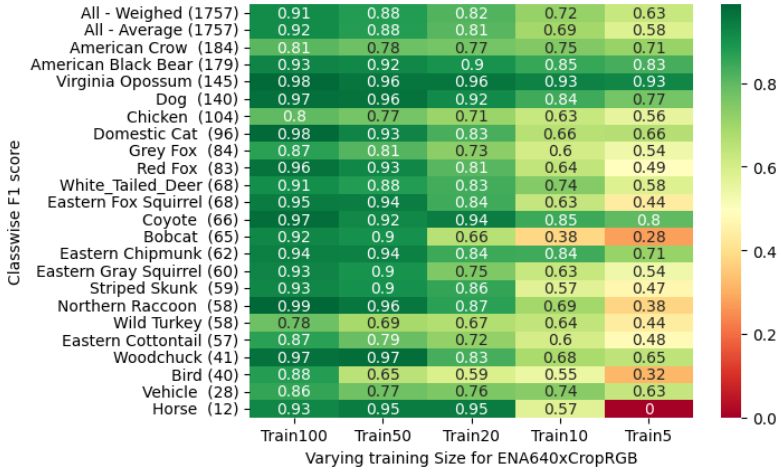


Figure 6.1: Influence of decreasing the number of training samples on performance
Yolov5s model, weighed classwise F1 score on the uncompressed ENA dataset in RGB. Support is indicated between the parenthesis for each class

without much loss of precision, recall, or F1 score. For example, looking at the Results Table (6.2), the F1 scores of training Yolov5s with the ENA640xCropGWNTrain100 subset, i.e., images of size 640x640 in gray-scale and compressed with "naive" and wavelet compression, as compared to the non-compressed subset in RGB, reduces the subset size from 909.3MB on disk down to 314.6MB on disk. However, the F1 score is mostly unchanged from 0.912 to 0.911. Compared to the *original* source images (i.e., with a resolution larger than 640x640, see Table 5.1), a more than 15 times reduction in the storage on disk is achieved. The fact that there is little effect on the classification capabilities of the model trained using the smallest subset amongst the compressed and gray-scale subsets is an exciting result, especially considering the edge computing domain.

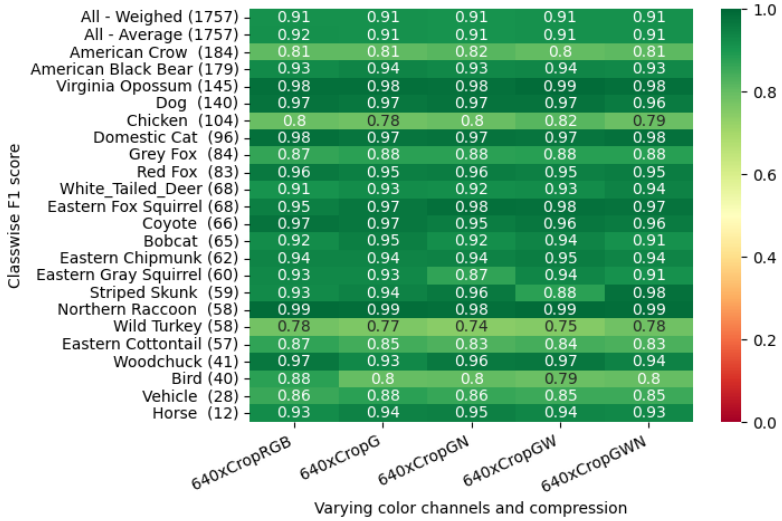


Figure 6.2: Influence of color channels and compression on the performance of the yolov5s model (ENA dataset)

6.1.3 Reduced image resolution and novel approach

This subsection described the results of reducing the image resolutions on the ENA24 dataset. It will first discuss results on the Yolov5s model, and later introduce results from our novel VAE-SVM-665k approach. The final four rows of Table 6.2 reduce the image dimensions in terms of image size, in addition to reducing the number of training samples for the ENA224xCropGWNTrain20 subset. The ENA224xCropGWNTrain20 subset is an interesting comparison to ENA64xCropGNTrain100 as they are similar in size on disk while being different in terms of dimensionality reduction. As is especially visible in the heatmap in Figure 6.4, the influence of reducing the image resolution is quite significant on the model performance. If we are only interested in the total training subset size on disk, the ENA224xCropGWNTrain20 training subset seems to be most promising in terms of model performance at an F1 score of 0.881 with only 53.4MB stored on disk (compared to 4.8GB (!!) of the original full-resolution training set). The Yolov5s model struggles with the subsets with images of 64x64 resolution. At the point of 64x64 images, the amount of information left in the resized images is very little compared to the original images. Smaller animals, for instance, rodents, are especially difficult to classify in this setting. An example of the size of these images was shown in Figure 6.3.

Figure 6.3: Image processed with different settings using the data pipeline. The 224x224 images showcase the difference between using the sequencing trick to detect and crop aroud an animal, and letterboxing the original image. The 64x64 image shows the size of the smallest images used the experiments

It is unexpected to see that Yolov5s performs as poorly in the 64x64 gray-scale image setting. The final two rows of Table 6.2 allow us to compare the performance of Yolov5s with our novel approach, VAE-SVM-665k. In the Table 6.3, we can closely compare the VAE-SVM-665k and

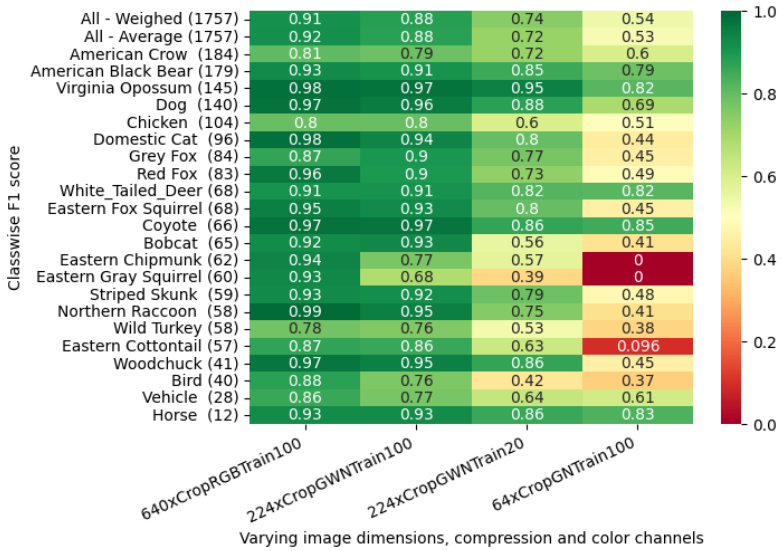


Figure 6.4: Influence of input dimensionality reduction on the performance of the yolov5s model (ENA dataset)

Yolov5s on a class-wise level. Excitingly, the VAE model is able to perform better in the overall (weighed and average) scores, and the VAE model scores significantly better on many of the rodent classes. The two rodent classes *Eastern Gray Squirrel (1)* and *Eastern Chipmunk (2)* have an F1 score of 0.775 and 0.926, respectively for the VAE model, whereas Yolov5s scores 0.000 for both. For the rodent class, *Eastern Fox Squirrel (12)* these values are 0.698 and 0.451 for the VAE and Yolov5s model. This demonstrates the superiority of the VAE model for the classes, which are small in size in the original images, and thus increasingly difficult to classify in the smaller images. This is likely due to the fact that the VAE model is able to better learn representations of the camera trap locations and the animals in the images needed to accurately classify the animals. It is worth noting that the Yolov5 model outperforms the classes, which are not well represented in the training subset. For instance, the VAE model has an F1 score of 0 for horses (a class that is not well represented in the dataset), whereas the Yolo5s model performs quite well with an F1 score of 0.831.

In summary, these results show that reducing the number of training images, the image sizes, and the other image dimensions through compression and gray-scale conversion can have a significant impact on the performance of the Yolov5 model on the ENA24 dataset. Unexpectedly, decreasing the number of training images does reduce the performance of Yolov5s, but interestingly certain classes are more affected by this than others. Most promisingly, the results show that compression and gray-scale conversion can be valuable techniques to reduce the required storage of wildlife data on disk, without significantly impacting performance. A significant reduction in the size of the training images on disk can be achieved while maintaining a high level of performance, up to 2.8 times from the largest images the model is trained with, and a reduction of up to 15 times as compared to the original source images. This is especially interesting for the edge computing domain, where the computational resources are limited and the amount of data that can be stored on the device is limited. The performance of Yolov5s was drastically impacted by reducing the image resolutions to 64x64, and promisingly, the novel VAE-SVM approach was able to achieve better results. This is true in particular for small animals such as rodents. On the other hand, Yolov5s performed better in classes that are not well represented in the dataset. Given that VAE-SVM-665k is more than 10 times smaller than Yolov5s in terms of trainable parameters, and more than 20 times smaller during inference time (since only the encoder is used during infer-

Table 6.3: Performance of VAE-SVM-665k versus Yolov5s on ENA64xCrop subset

		VAE-SVM-665K			Yolov5s		
Class	support	precision	recall	f1-score	precision	recall	f1-score
All - Average	1757	65.36%	55.41%	**0.581**	64.67%	44.34%	0.497
All - Weighed	1757	67.38%	63.69%	**0.634**	65.06%	49.94%	0.540
Bird (0)	40	75.00%	52.50%	**0.618**	70.20%	25.00%	0.369
Eastern Gray Squirrel (1)	60	97.37%	61.67%	**0.755**	0.00%	0.00%	0.000
Eastern Chipmunk (2)	62	94.92%	90.32%	**0.926**	0.00%	0.00%	0.000
Woodchuck (3)	41	100.00%	60.98%	**0.758**	56.40%	37.90%	0.453
Wild Turkey (4)	58	80.95%	29.31%	**0.430**	42.70%	34.50%	0.382
White_Tailed_Deer (5)	68	47.00%	69.12%	0.560	80.20%	83.80%	**0.820**
Virginia Opossum (6)	145	86.59%	97.93%	**0.919**	80.10%	83.40%	0.817
Eastern Cottontail (7)	57	41.67%	17.54%	**0.247**	38.40%	5.49%	0.096
American Black Bear (8)	179	41.02%	76.54%	0.534	81.20%	76.50%	**0.788**
Vehicle (9)	28	48.15%	46.43%	0.473	90.70%	46.40%	**0.614**
Striped Skunk (10)	59	48.08%	42.37%	0.450	58.30%	40.20%	**0.476**
Red Fox (11)	83	44.19%	45.78%	**0.450**	52.20%	46.00%	**0.489**
Eastern Fox Squirrel (12)	68	73.77%	66.18%	**0.698**	58.10%	36.80%	0.451
Northern Raccoon (13)	58	82.76%	41.38%	**0.552**	58.60%	31.00%	0.405
Grey Fox (14)	84	76.19%	57.14%	**0.653**	51.30%	40.20%	0.451
Horse (15)	12	0.00%	0.00%	0.000	93.10%	75.00%	**0.831**
Dog (16)	140	82.20%	69.29%	**0.752**	76.50%	62.90%	0.690
American Crow (17)	184	67.39%	84.24%	**0.749**	59.50%	61.40%	0.604
Chicken (18)	104	85.86%	81.73%	**0.837**	60.70%	43.30%	0.505
Domestic Cat (19)	96	54.79%	41.67%	**0.473**	52.60%	37.50%	0.438
Coyote (20)	66	82.93%	51.52%	0.636	91.30%	79.10%	**0.848**
Bobcat (21)	65	27.06%	35.38%	0.307	70.60%	29.20%	**0.413**

ence), these results are particularly noteworthy. Overall, these findings are promising for efforts to improve the feasibility of classification on the edge in the wildlife domain.

6.2 Performance evaluation of GhostNet and its variants

In the previous section, we have shown the impact of dimensionality reduction on the performance of the Yolov5s model as well as the performance of the VAE-SVM in comparison to Yolov5s. In this section, we will explore the alternative techniques to improve the performance of the classification on the edge using the GhostNet model and its variants and using Yolov5s as a benchmark. The previous experiments were run on the ENA24 dataset[61], and the experiments discussed in this section are run on the Channel Islands dataset [18]. The reason for this is that the Channel Islands dataset contains images taken in sequence. These images taken in sequence can be pre-processed with the 'sequencing' trick described in the Methodology Section 4.2. Table 6.4 is a collection of results from classifying images from the channel islands dataset [18] cropped to 224x224 dimensions in RGB. The first four rows showcase the GhostNet experiments. The full results with the class-wise performance of these experiments can be found in Table 6.5. The full class-wise results of the other Channel Islands dataset experiments can be found in Appendix Tables A.8 and A.9.

Table 6.4: Summary of experimental results of GhostNet experiments with Channel Islands training data.

Model	**Subset name**	**Precision**	**Recall**	**F1**
Dual GhostNet	ISL224xCrop/SeqRGBTrain5	70.62%	69.18%	0.687
GhostNet	ISL224xSeqRGBTrain5	74.37%	72.98%	0.732
GhostNet	ISL224xCropRGBTrain5	69.67%	68.36%	0.659
GhostNet-Hist-LBP	ISL224xCropRGBTrain5	68.56%	68.99%	0.684
Yolov5s	ISL224xCropRGBTrain5	43.69%	48.62%	0.432
Yolov5s	ISL224xSeqRGBTrain5	**81.43%**	**81.88%**	**0.810**

Table 6.5 shows the class-wise performance of the various GhostNet experiments in isolation. A common theme in all class-wise results (also seen in Appendix Tables A.8 and A.9) is that the

classes "Skunk", "Bird" and "Other" perform poorly compared to the rest. This can be explained by the fact that the number of images in the dataset for these classes is quite low. Comparing the GhostNet experiments, surprisingly, the unaltered GhostNet model outperforms both the Dual GhostNet and the Color Histogram + Local Binary Patterns GhostNet variant. The Dual GhostNet combines the input of the 'sequenced' and regularly cropped images. For the Color Histogram and the Local Binary Pattern approach it means that the model is either not able to effectively learn a relationship between the extracted features and the classes, or that the color histogram and local binary pattern features are not providing the model with new information with which classification can be improved. In terms of the Dual GhostNet approach, it is strange that the performance is poorer than the model which only takes in the cropped or sequenced images. This could be explained by a model architecture that does not perform well with the dual input approach, or that the information gained by using both of the images as input is not significant enough to yield better results.

Most interestingly, the models perform significantly better on the sequenced images than on the cropped images. In the case of the benchmark model Yolov5s, this result is especially visible. In the summary Table 6.4, using the sequenced images for training and testing yielded an increase from 0.432 to 0.810 in F1 score. Furthermore, Appendix Table A.8 shows that Yolov5s was not able to classify any rodents in the cropped subset, but reached an F1 score of 0.838 for the rodent class in the sequenced subset. This indicates that preprocessing images taken in sequence to generate bounding boxes provide valuable information to models with relatively low input resolution (in this case 224x224). This seems to be especially true for Yolov5s which is a two-stage model, first detecting an object and classifying afterward. Further improvements and applications of applying the sequence trick in the context of models with low input resolutions could yield even better results and improve the robustness of wildlife classification on the edge when applied correctly.

Table 6.5: Performance of GhostNet variants trained on ISL224xCropTrain5 and ISL224xSeqTrain5 data

		Dual ghostnet: crop. & seq.			GhostNet: Crop.			GhostNet: Seq.			GhostNet: Crop., Seq., CHist., LBP		
Class	support	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	8593	70.62%	69.18%	0.687	69.67%	68.36%	0.659	74.37%	72.98%	**0.732**	68.56%	68.99%	0.684
Empty (0)	3981	89.15%	69.41%	0.781	78.72%	75.76%	0.772	77.32%	82.65%	0.799	75.61%	85.10%	**0.801**
Other (1)	2	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Fox (2)	1831	48.55%	75.34%	0.591	24.58%	68.18%	0.361	60.29%	68.83%	**0.643**	66.96%	49.36%	0.568
Skunk (3)	24	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Rodent (4)	2544	64.07%	70.84%	0.673	94.46%	63.29%	0.758	86.83%	67.64%	**0.760**	65.06%	64.32%	0.647
Bird (5)	212	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000

6.3 Deeper performance evaluation of VAE-SVM

This section delves deeper into the performance evaluation of the novel VAE-SVM model approach. We discuss the reduction of the VAE-SVM model size by reducing size of the internal layers. This is followed by more comparison to our Yolov5 benchmark model in terms of performance and inference speed, and lastly, visualizations and analysis of the VAE latent space using t-SNE.

6.3.1 VAE-SVM-665k versus VAE-SVM-4M

We had originally intended to compare VAE-SVM-4M with models of similar sizes, such as Yolov5s (7.2M) and GhostNet (3.9M) (see Table 6.1). However, in a later stage of the research, it became clear that it was possible to further reduce the size of the VAE architecture by reducing the number- and breadth of the internal layers. This did not impact the performance of the VAE-SVM models significantly, as can be seen in Appendix Tables A.7 and A.9. From the research objectives perspective, it is highly beneficial for the edge domain to reduce the number of parameters without a significant impact on the performance. As many of the experiments were already

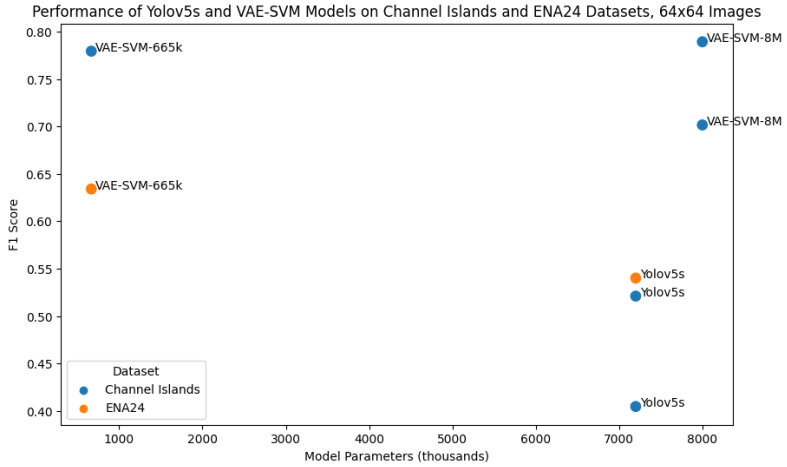


Figure 6.5: VAE-SVM outperforms Yolov5s on the 64x64 subsets while having fewer trainable parameters

run at this point in the research, a combination of VAE-SVM-665k and VAE-SVM-4M results have been shown. However, most experiments were run twice, once with VAE-SVM-4M and once with VAE-SVM-665k. The SVM is identical in both the VAE-SVM-4M and VAE-SVM-665k.

6.3.2 VAE-SVM-665k outperforms Yolov5s

We have previously seen that the VAE-SVM-665k model outperforms Yolov5s on the ENA24 dataset using the most constrained input data, ENA64xCropGNTrain100 (Table 6.2). In Table 6.6 we see that in addition to outperforming Yolov5s on the constrained ENA24 subset, the VAE-SVM approach is also outperforming Yolov5s on the Channel Islands 64x64 subset. In Figure 6.5 we visualize these results and highlight the performance of the VAE-SVM models and Yolov5s in comparison to the number of trainable parameters.

Interestingly, Yolov5s performs better on the "sequenced" subset than the cropped subset, but this is not the case for the VAE-SVM model. This suggests that Yolov5s could have trouble finding an appropriate bounding box in the 64x64 models, which is in agreement with previous research: in the study of Leorna et al. [37], they describe that the minimum detection limit for an animal is around 60 pixels using MegaDetector, which also takes a two-stage approach of detecting and classifying. Applying the sequencing trick to images could bypass this minimum detection limit as the objects in the image remain much larger, which results in improved performance.

Comparing both results from the 64x64 ENA24 subset (Table 6.2) and Channel Islands subset (Table 6.6), the performance difference between the VAE-SVM-665k and Yolov5s models on the ENA dataset is less than on the Channel Islands dataset. This can be attributed in part to the fact that Yolov5s performs quite poorly on small animal classes such as rodents, whereas VAE-SVM-665k has less difficulty with these classes. The fraction of rodents in the Channel Islands dataset is much higher in the Channel Islands dataset than in the ENA dataset.

6.3.3 Yolov5s and VAE-SVM-665k inference speed

Table 6.7 compares the inference speed per image of Yolov5s and VAE-SVM-665k. The results are from running experiments on a local machine, running on CUDA (NVIDIA GTX 1070) and an

Table 6.6: Summary of experimental results of Yolov5s and VAE-SVM on the 64x64 Channel Islands subset

Model	Subset name	Precision	Recall	F1
VAE-SVM-4M	ISL64xCropRGBTrain5	**78.64%**	**79.91%**	**0.789**
VAE-SVM-4M	ISL64xSeqRGBTrain5	71.04%	71.58%	0.702
VAE-SVM-665k	ISL64xCropRGBTrain5	77.42%	79.01%	0.779
Yolov5s	ISL64xCropRGBTrain5	37.01%	44.61%	0.405
Yolov5s	ISL64xSeqRGBTrain5	44.79%	62.27%	0.521

AMD Ryzen 7 1700x. Both of the models were tested on RGB images from the Channel Islands dataset, in a batch of 32 (i.e. input tensors of 32x3x64x64). For the VAE-SVM-665k model, the majority ($\geq$99%) can be attributed to the classification of the latent representations of the images using the SVM, whereas the encoding of the images took only 0.003 *milliseconds* per image. It is also worth noting that the Yolov5s model's inference speed does not include the time required for Non-Max Suppression (NMS), which can add an additional 1.4-1.7ms per image. This is left out as we are comparing classification results, and NMS is related to selecting the most appropriate bounding box. These results suggest that the VAE-SVM-665k may be more efficient in resource-constrained environments such as wildlife classification on the edge.

Table 6.7: Comparing inference speed between Yolov5s and VAE-SVM-655k

Model	Trainable Parameters	Inference speed
Yolov5s	7.2M	0.5ms-0.8ms*
VAE-SVM-665k	665K	0.3ms

* This excludes Non Max Suppresion (NMS) time of 1.4-1.7ms per image

6.3.4 Comparing latent space representations

By using the t-SNE algorithm, we can project the latent space embeddings of images after passing them through the encoder model as part of the VAE. Figures 6.8, 6.6 and 6.7 show these embeddings. The colors represent the different classes. The projection of the ENA24 dataset suggests that the encoder is able to generate embeddings, which are somewhat separable but not entirely. This is reflected in the classification results: The SVM was not fully capable of finding feasible separation boundaries in the latent space. For the Channel Islands subsets, it seems from visual inspection, that the embeddings of the sequenced channel island images are less separable than the cropped images. Unsurprisingly, Table 6.6 reflects this in the F1 scores, as the VAE-SVM F1 score on the cropped subset is higher than that of the sequenced subset.

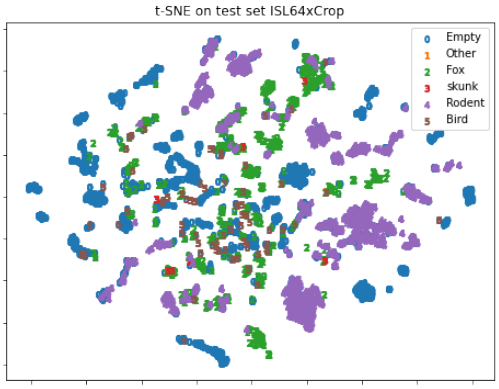


Figure 6.6: 2d projection of the Channel Islands test set (Cropped) embeddings using t-SNE (VAE-SVM-4M)

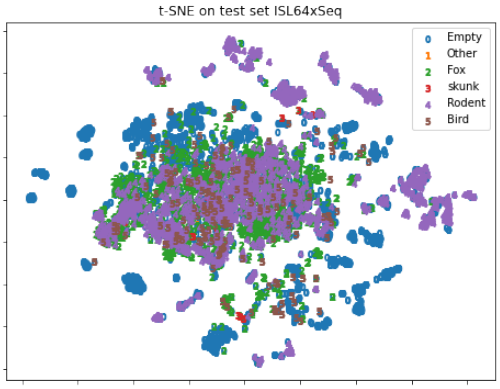


Figure 6.7: 2d projection of the Channel Islands test set (Sequenced) embeddings using t-SNE (VAE-SVM-4M)

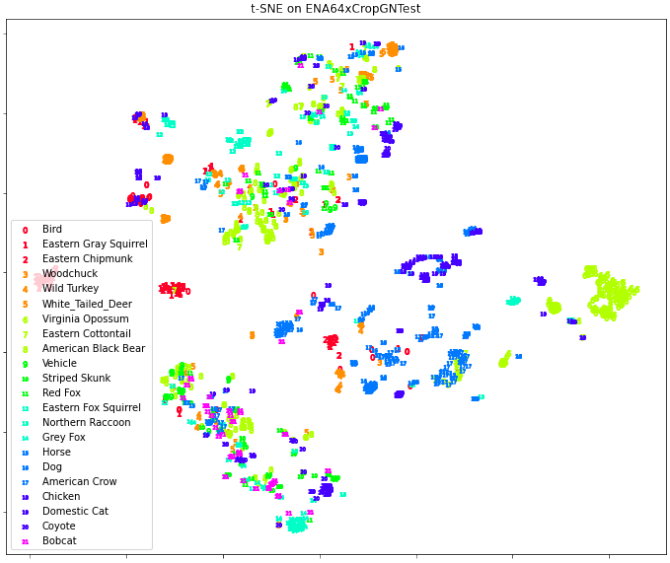


Figure 6.8: 2d projection of the ENA 64x64 test set embeddings using t-SNE (VAE-SVM-4M)

Chapter 7

Conclusion and Future Work

We set out to answer the question *which AI model and data design choices can enable edge-based wildlife image classification.* To reiterate the reasoning behind choosing this research question, our goal is to increase the viability of wildlife classification on the edge. We attempt to do so by addressing the research questions which address gaps both in domain research, as well aid in bringing the classification of wildlife on the edge to practice. In this chapter, we highlight what we have achieved, how our results can aid to answer these research questions, and what we can conclude about the research.

Encapsulating the first sub-question *Which design choices in reducing input image dimensionality can enable edge-based animal monitoring using AI on the edge?*, we started by investigating the impact of reducing the dimensions of deep learning models for animal classification on the edge during training and inference time. These experiments were run on the ENA24 dataset and used Yolov5s as a benchmark model. First, the number of training images was reduced, followed by gray-scale conversion and compression of the images, and lastly reducing the image resolutions from 640x640 down to 64x64. Unsurprisingly, reducing the number of training images results in a decrease in F1 score, precision, and recall. Classes with less initial support suffered more from the reduction in training images than other classes. Most promisingly, we have shown that the storage required for training images can be significantly reduced. The extent of this is that there is no significant impact on the performance while reducing the size of the images on disk by three times compared to their unaltered 640x640 versions, and up to 15 times compared to the images in original resolution. This reduction is pushed to a nearly 90 times reduction (53.4MB compared to 4.8GB) by combining gray-scale conversion, compression, and image resolution reduction, at which point the F1 score suffers a decrease from 0.913 to 0.881. The significant reduction in storage requirements for camera trap images can prove to be a significant factor in increasing the viability of animal monitoring on the edge. Increasing the capacity of storing or transmitting images used for training models in edge scenarios is key, where such resources are scarce. In conclusion to our first research question, by applying a combination of gray-scale conversion, compression, and resolution reduction techniques, wildlife classification on the edge can be made more feasible.

Our second sub-question reads *How can we design wildlife classification edge models tailored for reduced input image dimensions?*. With images heavily reduced in dimensions, we have set out to design a model capable of dealing with these constrained inputs more robustly than our benchmark model, Yolov5s. The proposed model is a combination of the Variational Autoencoder (VAE) and a Support-Vector-Machine (SVM). The VAE is trained (unsupervised) on training data, after which the SVM is trained to classify images that are encoded in the latent space of the VAE. This approach has proved surprisingly effective; for both the ENA24 dataset and the Channel Islands dataset experiments, the VAE outperformed the Yolov5s model on the most constrained subsets. These most constrained subsets contain images reduced to resolutions of 64x64, and in the case of ENA24, compressed and converted gray-scale. Both models perform quite poorly, but VAE shows

significant improvements. In particular, images with small animals were mostly not able to be classified to any extent by Yolov5 at the 64x64 resolution, but the VAE-SVM approach was able to classify these images surprisingly well in both the ENA24 and Channel Islands datasets. The VAE-SVM approach did struggle with classes that are not well represented in the training data. Comparing the size of Yolov5s with 7.2M trainable parameters, and VAE-SVM-665k with 665k trainable parameters, the VAE-SVM model is able to outperform the benchmark model with significantly fewer parameters. Yolov5s has an F1 score of 0.521 on the 64x64 Channel Islands dataset which made use of the *sequencing* trick, and the VAE-SVM-665k model showed an accuracy of 0.789 on the non-sequenced images. For the ENA24 dataset, Yolov5s achieved an F1 score of 0.540, while the VAE-SVM-665k model outperformed it with an F1 score of 0.634. Consequently, we answer this sub-question by proposing the VAE-SVM model design tailored for wildlife classification on the edge given constrained input data. With further research and refinement, the VAE-SVM model architecture approach could prove to be a valuable tool in wildlife classification on the edge.

Addressing the third research sub-question, *To what extent can alternative and non-traditional imaging techniques increase the classification results of the edge models?*, two types of alternative approaches are considered throughout the experiments. First, the two a-priori descriptors, color histograms, and local binary patterns. We experiment with extracting the color histograms and local binary patterns as features from the wildlife imagery during the date pipeline step. The reasoning behind using the combination of the color and texture descriptor is that they could prove to be a computationally efficient manner of extracting features from (full resolution) images, which can aid lightweight models coping with the reduced resolution of the input images. However, the results of the GhostNet variant utilizing the color histogram and local binary patterns as additional inputs did not yield any improvements over the GhostNet variant classifying on just the image. The reason for this is ambiguous, but we can hypothesize about two angles; first the color histograms and local binary did not yield valuable new information to the model. From the color histogram perspective, this would be in line with previous research suggesting that night- and daytime camera imagery (i.e., the availability of color) barely influences wildlife classification performance [52]. On the other hand, our model architecture or implementation could be lacking. While the results of the Color Histograms and Local Binary Patterns were not particularly interesting, the second alternative approach showed more promise. It works by taking a sequence of captured images, removing the background, and cropping around the object which remains. Hereby, we can crop around an animal that moves through the frame. The reasoning behind choosing this technique is that in images with very low resolution, small animals such as rodents are sized down to a few pixels making them hard to classify. By preemptively applying this *sequencing trick* to full-resolution images, we have been able to improve the performance of our models in certain cases. In particular, the performance of the Yolo models was increased significantly by applying the sequencing trick. In the 224x224 Channel Islands subset, the F1 score of Yolov5s on the images resized using letterboxing was 0.432, and the F1 score of the Yolov5s model on the images with the sequencing trick applied was close to double at 0.810. For the 64x64 images of the same Channel Islands dataset, the difference was less pronounced, jumping from an F1 score of 0.405 for the regular images, to 0.521 for the images with the sequencing trick. Surprisingly, the VAE-SVM model seems to perform better with the non-sequenced images. We believe that these results can be explained by that Yolov5s is able to better find and classify objects in sequenced images. These results suggest that alternative and non-traditional imaging techniques have their place in wildlife classification models on the edge, in particular when a technique is suited for a specific scenario or setting.

In conclusion, we have shown that application of a variety of input dimensionality reduction techniques can significantly reduce the amount of training data required without significantly sacrificing model performance. Second, we have proposed a VAE-SVM model design as a tailored approach for dealing with constrained inputs in the wildlife classification domain which outperforms existing models in low-resolution imagery, while using fewer model parameters. Furthermore, our experiments on non-traditional and alternative techniques for wildlife classification on the edge

suggest that there lies potential in the practical use of techniques such as the sequencing trick. Ultimately, these findings contribute to the development and feasibility of wildlife classification on the edge.

7.1 Future research

In this section, we address potential future research opportunities.

Previous research indicated that wildlife images taken at nighttime did not influence wildlife classification performance [52]. Our research further builds upon this by showing that images can be compressed, reduced in size, and turned gray-scale to significantly reduce storage requirements while retaining the performance of models trained and evaluated using these images. Future research can build upon this, for example through the use of different compression techniques or color spaces, and by developing models and exploring applications that benefit most from these optimized image representations.

The VAE-SVM model has only been applied to the most constricted datasets. There exist opportunities to explore the VAE-SVM model outside of the context of the constrained input image and the edge, as well as outside of the wildlife domain. One approach we see as applicable is using a chunking approach. By analyzing a high-resolution image chunk-by-chunk, the VAE-SVM model can classify each of these lower-resolution chunks separately. It might be required to train a VAE for each of the chunks, but this is possible given that the VAE model can be trained using unsupervised learning. The SVM classifier would only have to be trained with a small number of labeled images, and one could feasibly apply an active learning approach to reduce the number of required labels for training. Other opportunities exist for applying the VAE-SVM approach, and we suggest exploring its potential in both practical and research settings.

In a setting such as Wildlife classification, it is valuable to perform research in collaboration with domain experts. After our preliminary results were ready, we had the opportunity to discuss the results with Dan Morris, who we previously introduced as the researcher involved with various camera trapping projects such as MegaDetector [11]. From his perspective, the sequencing trick could be particularly useful in certain scenarios where small animals cannot be detected by object detection. More concretely, the MegaDetector model performs well in detecting large animals in various settings but can have difficulty detecting smaller animals, such as rodents, passing by the camera trap at high speeds. In our results, we have shown that applying the sequencing trick to images allows for the detection of animals that object detection models had difficulty finding, and as such can be explored to aid in the scenario mentioned. Furthermore, the sequencing trick could be applied as an early filter for removing ghost images. In an informal experimental setting, the images for which a bounding box around the animal was not found using the sequencing trick, and a large portion of the images for which no bounding box was found were empty or 'ghost' images. Consequently, we suggest further exploring and fine-tuning the sequencing approach for the wildlife classification domain and beyond.

Bibliography

[1] GitHub - huawei-noah/Efficient-AI-Backbones. https://github.com/huawei-noah/Efficient-AI-Backbones, September 2022. 31

[2] GitLab - camelot Project. https://gitlab.com/camelot-project/camelot, August 2022. 6

[3] Animl: A frontend web app for viewing & labeling camera trap data by the nature conservancy. https://www.animl.org/, March 2023. 6

[4] GitHub - AntixK/PyTorch-VAE: A Collection of Variational Autoencoders (VAE) in PyTorch. https://github.com/AntixK/PyTorch-VAE, March 2023. 32

[5] TrapTagger: AI-Powered Camera-Trap Imagery Processing. https://wildeyeconservation.org/traptagger/, April 2023. 6

[6] Jorge A. Ahumada, Eric Fegraus, Tanya Birch, Nicole Flores, Roland Kays, Timothy G. O'Brien, Jonathan Palmer, Stephanie Schuttler, Jennifer Y. Zhao, Walter Jetz, Margaret Kinnaird, Sayali Kulkarni, Arnaud Lyet, David Thau, Michelle Duong, Ruth Oliver, and Anthony Dancer. Wildlife Insights: A Platform to Maximize the Potential of Camera Trap and Other Passive Sensor Wildlife Data for the Planet. *Environmental Conservation*, 47(1):1–6, March 2020. Publisher: Cambridge University Press. 6

[7] Vaneeda Allken, Nils Olav Handegard, Shale Rosen, Tiffanie Schreyeck, Thomas Mahiout, and Ketil Malde. Fish species identification using a convolutional neural network trained on synthetic data. *ICES Journal of Marine Science*, 76(1):342–349, January 2019. 5

[8] Bilal Arshad, Johan Barthelemy, Elliott Pilton, and Pascal Perez. Where is my Deer?-Wildlife Tracking And Counting via Edge Computing And Deep Learning. In *2020 IEEE SENSORS*, pages 1–4, October 2020. ISSN: 2168-9229. 7

[9] Daphne Auer, Paul Bodesheim, Christian Fiderer, Marco Heurich, and Joachim Denzler. Minimizing the Annotation Effort for Detecting Wildlife in Camera Trap Images with Active Learning. January 2021. 5

[10] M. Mahdi Azari, Sourabh Solanki, Symeon Chatzinotas, Oltjon Kodheli, Hazem Sallouha, Achiel Colpaert, Jesus Fabian Mendoza Montoya, Sofie Pollin, Alireza Haqiqatnejad, Arsham Mostaani, Eva Lagunas, and Bjorn Ottersten. Evolution of Non-Terrestrial Networks From 5G to 6G: A Survey. *IEEE Communications Surveys & Tutorials*, 24(4):2633–2672, 2022. 6

[11] Sara Beery, Dan Morris, and Siyu Yang. Efficient pipeline for camera trap image review. *arXiv preprint arXiv:1907.06772*, 2019. 1, 3, 6, 13, 24, 28, 29, 45

[12] Francesco Bianconi, Antonio Fernández, Fabrizio Smeraldi, and Giulia Pascoletti. Colour and Texture Descriptors for Visual Recognition: A Historical Overview. *Journal of Imaging*, 7(11):245, November 2021. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute. 12

[13] Paul Bodesheim, Jan Blunk, Matthias Körschens, Clemens-Alexander Brust, Christoph Käding, and Joachim Denzler. Pre-trained models are not enough: active and lifelong learning is important for long-term visual monitoring of mammals in biodiversity research—Individual identification and attribute prediction with image features from deep neural networks and decoupled decision models applied to elephants and great apes. *Mammalian Biology*, April 2022. 2, 5

[14] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 18, 19, 21

[15] Otto Brookes, Majid Mirmehdi, Hjalmar Kühl, and Tilo Burghardt. Triple-stream Deep Metric Learning of Great Ape Behavioural Actions, January 2023. arXiv:2301.02642 [cs]. 3, 5

[16] Gerardo Ceballos, Paul R. Ehrlich, and Peter H. Raven. Vertebrates on the brink as indicators of biological annihilation and the sixth mass extinction. *Proceedings of the National Academy of Sciences*, 117(24):13596–13602, June 2020. Publisher: Proceedings of the National Academy of Sciences. 1

[17] Guobin Chen, Tony X. Han, Zhihai He, Roland Kays, and Tavis Forrester. Deep convolutional neural network based species recognition for wild animal monitoring. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 858–862, October 2014. ISSN: 2381-8549. 5

[18] The Nature Conservancy. Channel islands camera traps 1.0, 2021. Dataset. 2, 28, 30, 38, 58, 63, 65

[19] Fagner Cunha, Eulanda M. dos Santos, Raimundo Barreto, and Juan G. Colonna. Filtering Empty Camera Trap Images in Embedded Systems, April 2021. arXiv:2104.08859 [cs]. 3, 6, 7

[20] Matthew Danish, Rohit Verma, Justas Brazauskas, Ian Lewis, and Richard Mortier. DeepDish on a diet: low-latency, energy-efficient object-detection and tracking at the edge. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, pages 43–48, Rennes France, April 2022. ACM. 8, 13

[21] J. P. Dominguez-Morales, A. Rios-Navarro, M. Dominguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, D. Cascado-Caballero, A. Jimenez-Fernandez, and A. Linares-Barranco. Wireless Sensor Network for Wildlife Tracking and Behavior Classification of Animals in Doñana. *IEEE Communications Letters*, 20(12):2534–2537, December 2016. Conference Name: IEEE Communications Letters. 8

[22] Alexander Gomez, German Diez, Augusto Salazar, and Angelica Diaz. Animal Identification in Low Quality Camera-Trap Images Using Very Deep Convolutional Neural Networks and Confidence Thresholds. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Fatih Porikli, Sandra Skaff, Alireza Entezari, Jianyuan Min, Daisuke Iwai, Amela Sadagic, Carlos Scheidegger, and Tobias Isenberg, editors, *Advances in Visual Computing*, Lecture Notes in Computer Science, pages 747–756, Cham, 2016. Springer International Publishing. 1, 3, 5, 6

[23] Alexander Gomez Villa, Augusto Salazar, and Francisco Vargas. Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks. *Ecological Informatics*, 41:24–32, September 2017. 1, 3, 5, 6

[24] M. Grooten, R. E. A. Almond, and WWF (Organization), editors. *Living planet report 2018: aiming higher*. WWF–World Wide Fund for Nature, Gland, Switzerland, 2018. OCLC: on1102512826. 1

[25] D. Gutierrez-Galan, Juan P. Dominguez-Morales, E. Cerezuela-Escudero, A. Rios-Navarro, R. Tapiador-Morales, M. Rivas-Perez, M. Dominguez-Morales, A. Jimenez-Fernandez, and A. Linares-Barranco. Embedded neural network for real-time animal behavior classification. *Neurocomputing*, 272:17–26, January 2018. 7

[26] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. GhostNet: More Features from Cheap Operations, March 2020. arXiv:1911.11907 [cs]. 13, 17, 25, 31

[27] Liang Jia, Ye Tian, and Junguo Zhang. Identifying Animals in Camera Trap Images via Neural Architecture Search. *Computational Intelligence and Neuroscience*, 2022:e8615374, February 2022. Publisher: Hindawi. 7

[28] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, November 2022. 13, 24, 31

[29] Orjan Johansson, Gustaf Samelius, Ewa Wikberg, Guillaume Chapron, Charudutt Mishra, and Matthew Low. Identification errors in camera-trap studies result in systematic population overestimation. *Scientific Reports*, 10:6393, April 2020. 1, 2

[30] Gustavo Kaneto. Python opencv background subtraction and bounding box. `https://stackoverflow.com/questions/60646384/python-opencv-background-subtraction-and-bounding-box`, March 12 2020. 19

[31] Jaskirat Kaur and Williamjeet Singh. Tools, techniques, datasets and application areas for object detection in an image: a review. *Multimedia Tools and Applications*, 81(27):38297–38351, November 2022. 6

[32] Holly K. Kindsvater, Nicholas K. Dulvy, Cat Horswill, Maria-José Juan-Jordá, Marc Mangel, and Jason Matthiopoulos. Overcoming the Data Crisis in Biodiversity Conservation. *Trends in Ecology & Evolution*, 33(9):676–688, September 2018. 1

[33] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022. arXiv:1312.6114 [cs, stat]. 10

[34] Michał Koziarski and Boguslaw Cyganek. Impact of Low Resolution on Image Recognition with Deep Neural Networks: An Experimental Study. *International Journal of Applied Mathematics and Computer Science*, 28:735–744, December 2018. 4, 8

[35] Matthew Kutugata, Jeremy Baumgardt, John A. Goolsby, and Alexis E. Racelis. Automatic Camera-Trap Classification Using Wildlife-Specific Deep Learning in Nilgai Management. *Journal of Fish and Wildlife Management*, 12(2):412–421, July 2021. 3

[36] Huawei Noah's Ark Lab. Efficient-ai-backbones. `https://github.com/huawei-noah/Efficient-AI-Backbones`, 2021. Accessed: March 28, 2023. 25

[37] Scott Leorna and Todd Brinkman. Human vs. machine: Detecting wildlife in camera trap images. *Ecological Informatics*, 72:101876, December 2022. 1, 8, 40

[38] He Li, Kaoru Ota, and Mianxiong Dong. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. *IEEE Network*, 32(1):96–101, January 2018. Conference Name: IEEE Network. 7

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context, February 2015. arXiv:1405.0312 [cs]. 29, 31

[40] Paul D. Meek, Guy A. Ballard, Jess Sparkes, Mark Robinson, Brad Nesbitt, and Peter J. S. Fleming. Camera trap theft and vandalism: occurrence, cost, prevention and implications for wildlife research and management. *Remote Sensing in Ecology and Conservation*, 5(2):160–168, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rse2.96. 1, 3

[41] Zhongqi Miao, Kaitlyn M. Gaynor, Jiayun Wang, Ziwei Liu, Oliver Muellerklein, Mohammad Sadegh Norouzzadeh, Alex McInturff, Rauri C. K. Bowie, Ran Nathan, Stella X. Yu, and Wayne M. Getz. Insights and approaches using deep learning to classify wildlife. *Scientific Reports*, 9(1):8137, May 2019. Number: 1 Publisher: Nature Publishing Group. 1, 3, 5

[42] Zhongqi Miao, Ziwei Liu, Kaitlyn M. Gaynor, Meredith S. Palmer, Stella X. Yu, and Wayne M. Getz. Iterative Human and Automated Identification of Wildlife Images. *Nature Machine Intelligence*, 3(10):885–895, October 2021. arXiv:2105.02320 [cs]. 6

[43] Neal R Haddaway and David Leclere. *Bending the curve of biodiversity loss*. Number 2020 in Living planet report. WWF, Gland, 2020. 1

[44] Hung Nguyen, Sarah J. Maclagan, Tu Dinh Nguyen, Thin Nguyen, Paul Flemons, Kylie Andrews, Euan G. Ritchie, and Dinh Phung. Animal Recognition and Identification with Deep Convolutional Neural Networks for Automated Wildlife Monitoring. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 40–49, October 2017. 5, 6

[45] M. S. Norouzzadeh, Anh M. Nguyen, M. Kosmala, A. Swanson, C. Packer, and J. Clune. Automatically identifying wild animals in camera trap images with deep learning. *ArXiv*, March 2017. 5, 6

[46] Mohammad Sadegh Norouzzadeh, Dan Morris, Sara Beery, Neel Joshi, Nebojsa Jojic, and Jeff Clune. A deep active learning system for species identification and counting in camera trap images. *Methods in Ecology and Evolution*, 12(1):150–161, 2021. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.13504. 1, 3, 5

[47] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59, January 1996. 12

[48] Emmanuel Okafor, Pornntiwa Pawara, Faik Karaaba, Olarik Surinta, Valeriu Codreanu, Lambert Schomaker, and Marco Wiering. Comparative study between deep learning and bag of visual words for wild-animal recognition. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, December 2016. 5, 6, 9

[49] Vigneshwaran Palanisamy and Nagulan Ratnarajah. Detection of Wildlife Animals using Deep Learning Approaches: A Systematic Review. In *2021 21st International Conference on Advances in ICT for Emerging Regions (ICter)*, pages 153–158, December 2021. ISSN: 2472-7598. 5

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 32

[51] Andrew Shepley, Gregory Falzon, Christopher Lawson, Paul Meek, and Paul Kwan. U-Infuse: Democratization of Customizable Deep Learning for Object Detection. *Sensors*, 21:2611, April 2021. 5

[52] Michael A. Tabak, Mohammad S. Norouzzadeh, David W. Wolfson, Steven J. Sweeney, Kurt C. Vercauteren, Nathan P. Snow, Joseph M. Halseth, Paul A. Di Salvo, Jesse S. Lewis, Michael D. White, Ben Teton, James C. Beasley, Peter E. Schlichting, Raoul K. Boughton, Bethany Wight, Eric S. Newkirk, Jacob S. Ivan, Eric A. Odell, Ryan K. Brook, Paul M. Lukacs, Anna K. Moeller, Elizabeth G. Mandeville, Jeff Clune, and Ryan S. Miller. Machine learning to classify animal species in camera trap images: Applications in ecology. *Methods in Ecology and Evolution*, 10(4):585–590, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.13120. 8, 44, 45

[53] Ting Feng Tan, Soo Siang Teoh, Jun Ee Fow, and Kin Sam Yen. Embedded human detection system based on thermal and infrared sensors for anti-poaching application. In *2016 IEEE Conference on Systems, Process and Control (ICSPC)*, pages 37–42, December 2016. 2

[54] Devis Tuia, Benjamin Kellenberger, Sara Beery, Blair R. Costelloe, Silvia Zuffi, Benjamin Risse, Alexander Mathis, Mackenzie W. Mathis, Frank van Langevelde, Tilo Burghardt, Roland Kays, Holger Klinck, Martin Wikelski, Iain D. Couzin, Grant van Horn, Margaret C. Crofoot, Charles V. Stewart, and Tanya Berger-Wolf. Perspectives in machine learning for wildlife conservation. *Nature Communications*, 13(1):792, February 2022. Number: 1 Publisher: Nature Publishing Group. 1

[55] Wikimedia Commons: user EugenioTL. Image: Variational autoencoder structure, 2021. 11

[56] Wikimedia Commons: user ihasb33r. Image: An odd-eyed cat, 2008. 12

[57] Wikimedia Commons: user Kyle McDonald. Image: T-sne embedding of mnist. using default parameters and the multicore t-sne implementation or leland mcinnes' umap implementation., 2017. 11

[58] Wikimedia Commons: user Muenteroman. Image: Rgb-histogram of the odd-eyed cat picture shown in the color histogram article on the english wikipedia. x-axis is rgb, y-axis is frequency., 2015. 12

[59] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020. arXiv:1907.08349 [cs]. 7

[60] Zhangmingjia Yin and Fucheng You. Animal image recognition based on convolutional neural network. In *Proceedings of the 2020 4th International Conference on Electronic Information Technology and Computer Engineering*, EITCE 2020, pages 545–549, New York, NY, USA, February 2021. Association for Computing Machinery. 3, 5

[61] Hayder Yousif, Roland Kays, and Zhihai He. Dynamic programming selection of object proposals for sequence-level animal species classification in the wild. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019. 2, 28, 29, 34, 38, 53, 63, 65

[62] Hayder Yousif, Jianhe Yuan, Roland Kays, and Zhihai He. Animal Scanner: Software for classifying humans, animals, and empty frames in camera trap images. *Ecology and Evolution*, 9(4):1578–1589, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ece3.4747. 3, 5

[63] Tingting Zhao, Xiaoli Yi, Zhiyong Zeng, and Tao Feng. MobileNet-Yolo based wildlife detection model: A case study in Yunnan Tongbiguan Nature Reserve, China. *Journal of Intelligent & Fuzzy Systems*, 41(1):2171–2181, January 2021. Publisher: IOS Press. 8

[64] Imran Zualkernan, Salam Dhou, Jacky Judas, Ali Reza Sajun, Brylle Ryan Gomez, and Lana Alhaj Hussain. An IoT System Using Deep Learning to Classify Camera Trap Images on the Edge. *Computers*, 11(1):13, January 2022. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute. 1, 2, 3, 5, 7

Appendix A

Appendix to the results

A.1 Class-wise ENA24 Dataset results

This appendix section contains class-wise results from the experiments run on the ENA24 dataset [61] Reported scores are results from classification on the test set.

Table A.1: Comparing Yolov5l to Yolov5s performance on the ENA dataset

		Y5sENA640xCropRGBTrain100			Y5lENA640xCropRGBTrain100		
	Support	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	1757	90.24%	92.64%	0.913	90.28%	94.85%	0.924
All - Average	1757	90.50%	92.60%	0.915	91.00%	94.60%	0.928
Bird (0)	40	83.90%	91.50%	0.875	80.10%	90.50%	0.850
Eastern Gray Squirrel (1)	60	91.90%	94.40%	0.931	91.60%	96.70%	0.941
Eastern Chipmunk (2)	62	98.20%	90.30%	0.941	99.90%	91.90%	0.957
Woodchuck (3)	41	95.60%	97.60%	0.966	96.40%	97.60%	0.970
Wild Turkey (4)	58	73.00%	83.70%	0.780	68.60%	82.80%	0.750
White_Tailed_Deer (5)	68	86.70%	95.70%	0.910	90.00%	95.60%	0.927
Virginia Opossum (6)	145	99.50%	97.20%	0.983	99.70%	97.20%	0.984
Eastern Cottontail (7)	57	81.30%	93.00%	0.868	81.80%	94.70%	0.878
American Black Bear (8)	179	92.60%	93.30%	0.929	92.50%	96.10%	0.943
Vehicle (9)	28	83.20%	88.40%	0.857	84.40%	92.90%	0.884
Striped Skunk (10)	59	96.30%	89.20%	0.926	99.00%	98.30%	0.986
Red Fox (11)	83	98.70%	93.80%	0.962	95.80%	95.20%	0.955
Eastern Fox Squirrel (12)	68	93.00%	97.10%	0.950	98.20%	100.00%	0.991
Northern Raccoon (13)	58	99.50%	98.30%	0.989	99.70%	100.00%	0.998
Grey Fox (14)	84	85.10%	88.20%	0.866	86.60%	92.10%	0.893
Horse (15)	12	93.30%	91.70%	0.925	100.00%	90.80%	0.952
Dog (16)	140	96.10%	98.60%	0.973	96.40%	98.60%	0.975
American Crow (17)	184	74.10%	88.00%	0.805	71.90%	91.80%	0.806
Chicken (18)	104	80.50%	78.80%	0.796	77.40%	85.70%	0.813
Domestic Cat (19)	96	98.90%	97.50%	0.982	98.90%	97.90%	0.984
Coyote (20)	66	96.40%	98.50%	0.974	95.40%	100.00%	0.976
Bobcat (21)	65	92.60%	92.30%	0.924	97.50%	95.40%	0.964

Table A.2: Results of training subset size reduction on ENA dataset using Yolov5s (1/2)

	Y5sENA640xCropRGBTrain100			Y5sENA640xCropRGBTrain50		
Class	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	90.24%	92.64%	0.913	86.96%	89.10%	0.879
All - Average	90.50%	92.60%	0.915	86.60%	88.40%	0.875
Bird (0)	83.90%	91.50%	0.875	63.00%	67.50%	0.652
Eastern Gray Squirrel (1)	91.90%	94.40%	0.931	89.50%	90.00%	0.897
Eastern Chipmunk (2)	98.20%	90.30%	0.941	98.20%	89.70%	0.938
Woodchuck (3)	95.60%	97.60%	0.966	95.30%	99.90%	0.975
Wild Turkey (4)	73.00%	83.70%	0.780	63.40%	75.90%	0.691
White_Tailed_Deer (5)	86.70%	95.70%	0.910	83.90%	91.80%	0.877
Virginia Opossum (6)	99.50%	97.20%	0.983	97.10%	95.20%	0.961
Eastern Cottontail (7)	81.30%	93.00%	0.868	75.50%	82.50%	0.788
American Black Bear (8)	92.60%	93.30%	0.929	91.70%	92.70%	0.922
Vehicle (9)	83.20%	88.40%	0.857	74.00%	81.20%	0.774
Striped Skunk (10)	96.30%	89.20%	0.926	92.80%	88.10%	0.904
Red Fox (11)	98.70%	93.80%	0.962	94.90%	90.30%	0.925
Eastern Fox Squirrel (12)	93.00%	97.10%	0.950	98.00%	91.20%	0.945
Northern Raccoon (13)	99.50%	98.30%	0.989	94.80%	96.60%	0.957
Grey Fox (14)	85.10%	88.20%	0.866	78.70%	83.80%	0.812
Horse (15)	93.30%	91.70%	0.925	97.90%	91.70%	0.947
Dog (16)	96.10%	98.60%	0.973	97.10%	96.00%	0.965
American Crow (17)	74.10%	88.00%	0.805	72.80%	83.20%	0.777
Chicken (18)	80.50%	78.80%	0.796	74.40%	78.80%	0.765
Domestic Cat (19)	98.90%	97.50%	0.982	89.60%	96.90%	0.931
Coyote (20)	96.40%	98.50%	0.974	93.70%	90.70%	0.922
Bobcat (21)	92.60%	92.30%	0.924	89.70%	90.80%	0.902

Table A.3: Results of training subset size reduction on ENA dataset using Yolov5s (2/2)

	Y5sENA640xCropRGBTrain20			Y5sENA640xCropRGBTrain10			Y5sENA640xCropRGBTrain5		
Class	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	84.85%	79.61%	0.819	75.55%	70.12%	0.718	66.67%	60.72%	0.629
All - Average	85.00%	77.30%	0.810	73.00%	66.00%	0.693	65.80%	52.10%	0.582
Bird (0)	69.60%	51.40%	0.591	60.30%	50.00%	0.547	43.00%	25.00%	0.316
Eastern Gray Squirrel (1)	78.60%	71.70%	0.750	69.50%	57.10%	0.627	66.20%	45.00%	0.536
Eastern Chipmunk (2)	90.70%	79.00%	0.844	94.60%	75.80%	0.842	81.40%	63.50%	0.713
Woodchuck (3)	85.80%	80.50%	0.831	68.10%	68.30%	0.682	61.80%	68.30%	0.649
Wild Turkey (4)	67.00%	66.50%	0.667	60.80%	67.20%	0.638	48.80%	39.70%	0.438
White_Tailed_Deer (5)	86.20%	80.90%	0.835	81.00%	68.90%	0.745	66.50%	51.50%	0.580
Virginia Opossum (6)	98.00%	93.80%	0.959	94.50%	92.40%	0.934	92.90%	92.40%	0.926
Eastern Cottontail (7)	74.00%	70.20%	0.720	65.00%	56.10%	0.602	61.40%	39.10%	0.478
American Black Bear (8)	90.40%	89.30%	0.898	88.60%	82.30%	0.853	86.80%	79.90%	0.832
Vehicle (9)	99.70%	60.70%	0.755	94.40%	60.40%	0.737	100.00%	46.30%	0.633
Striped Skunk (10)	95.10%	78.00%	0.857	89.20%	42.10%	0.572	62.60%	37.30%	0.467
Red Fox (11)	91.90%	72.30%	0.809	72.40%	57.00%	0.638	47.30%	51.80%	0.494
Eastern Fox Squirrel (12)	85.70%	82.40%	0.840	60.10%	66.30%	0.630	47.20%	40.80%	0.438
Northern Raccoon (13)	93.50%	81.00%	0.868	75.70%	63.80%	0.692	43.70%	32.80%	0.375
Grey Fox (14)	73.60%	72.60%	0.731	54.50%	67.90%	0.605	52.50%	55.30%	0.539
Horse (15)	100.00%	90.50%	0.950	56.60%	58.30%	0.574	0.00%	0.00%	0
Dog (16)	96.10%	87.70%	0.917	90.20%	78.60%	0.840	75.40%	77.90%	0.766
American Crow (17)	71.60%	82.60%	0.767	69.30%	80.80%	0.746	65.80%	77.30%	0.711
Chicken (18)	77.30%	65.40%	0.709	65.40%	60.10%	0.626	60.70%	52.00%	0.560
Domestic Cat (19)	89.50%	77.10%	0.828	80.50%	55.90%	0.660	79.60%	56.90%	0.664
Coyote (20)	93.10%	95.50%	0.943	88.40%	81.20%	0.846	79.60%	80.30%	0.799
Bobcat (21)	61.80%	70.80%	0.660	27.70%	61.50%	0.382	23.90%	32.30%	0.275

Table A.4: Influence of image compression and grayscale conversion on the performance of the Yolov5s model (1/2)

	Y5sENA640xCropRGBTrain100			Y5sENA640xCropGTrain100		
	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	90.24%	92.64%	0.913	90.55%	91.97%	0.912
All - Average	90.50%	92.60%	0.915	91.00%	91.40%	0.912
Bird (0)	83.90%	91.50%	0.875	77.40%	82.50%	0.799
Eastern Gray Squirrel (1)	91.90%	94.40%	0.931	91.90%	94.20%	0.930
Eastern Chipmunk (2)	98.20%	90.30%	0.941	99.70%	88.70%	0.939
Woodchuck (3)	95.60%	97.60%	0.966	92.80%	93.90%	0.933
Wild Turkey (4)	73.00%	83.70%	0.780	76.10%	77.00%	0.765
White_Tailed_Deer (5)	86.70%	95.70%	0.910	90.20%	94.90%	0.925
Virginia Opossum (6)	99.50%	97.20%	0.983	99.30%	97.20%	0.982
Eastern Cottontail (7)	81.30%	93.00%	0.868	81.90%	87.40%	0.846
American Black Bear (8)	92.60%	93.30%	0.929	92.40%	95.60%	0.940
Vehicle (9)	83.20%	88.40%	0.857	89.90%	85.70%	0.877
Striped Skunk (10)	96.30%	89.20%	0.926	96.50%	92.20%	0.943
Red Fox (11)	98.70%	93.80%	0.962	97.90%	92.80%	0.953
Eastern Fox Squirrel (12)	93.00%	97.10%	0.950	96.20%	97.10%	0.966
Northern Raccoon (13)	99.50%	98.30%	0.989	100.00%	97.40%	0.987
Grey Fox (14)	85.10%	88.20%	0.866	85.00%	90.50%	0.877
Horse (15)	93.30%	91.70%	0.925	97.20%	91.70%	0.944
Dog (16)	96.10%	98.60%	0.973	96.50%	97.10%	0.968
American Crow (17)	74.10%	88.00%	0.805	75.70%	86.40%	0.807
Chicken (18)	80.50%	78.80%	0.796	78.00%	78.80%	0.784
Domestic Cat (19)	98.90%	97.50%	0.982	96.60%	97.90%	0.972
Coyote (20)	96.40%	98.50%	0.974	95.20%	98.50%	0.968
Bobcat (21)	92.60%	92.30%	0.924	96.30%	93.80%	0.950

Table A.5: Influence of image compression and grayscale conversion on the performance of the Yolov5s model (2/2)

	Y5sENA640xCropGNTrain100			Y5sENA640xCropGWTrain100			Y5sENA640xCropGWNTrain100		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	90.90%	91.31%	0.910	89.81%	92.97%	0.912	89.51%	93.03%	0.911
All - Average	91.20%	90.70%	0.909	90.00%	92.30%	0.911	89.80%	92.60%	0.912
Bird (0)	78.50%	82.40%	0.804	76.00%	82.50%	0.791	75.60%	85.00%	0.800
Eastern Gray Squirrel (1)	91.30%	83.30%	0.871	92.00%	95.50%	0.937	90.20%	92.40%	0.913
Eastern Chipmunk (2)	100.00%	89.10%	0.942	100.00%	91.40%	0.955	98.00%	90.30%	0.940
Woodchuck (3)	97.50%	94.80%	0.961	97.50%	96.10%	0.968	96.00%	92.70%	0.943
Wild Turkey (4)	74.60%	74.10%	0.743	70.90%	79.80%	0.751	74.80%	81.80%	0.781
White_Tailed_Deer (5)	90.00%	94.10%	0.920	89.10%	96.10%	0.925	90.20%	98.50%	0.942
Virginia Opossum (6)	98.90%	97.20%	0.980	100.00%	98.00%	0.990	98.30%	97.90%	0.981
Eastern Cottontail (7)	79.30%	87.40%	0.832	78.70%	90.60%	0.842	76.90%	91.20%	0.834
American Black Bear (8)	92.30%	93.70%	0.930	93.30%	93.80%	0.935	91.90%	94.50%	0.932
Vehicle (9)	83.20%	89.30%	0.861	85.20%	85.70%	0.854	85.00%	85.70%	0.853
Striped Skunk (10)	99.70%	93.20%	0.963	90.90%	84.80%	0.877	100.00%	95.60%	0.978
Red Fox (11)	97.20%	94.00%	0.956	97.70%	91.60%	0.946	96.90%	94.00%	0.954
Eastern Fox Squirrel (12)	99.80%	97.10%	0.984	97.50%	98.50%	0.980	96.20%	98.50%	0.973
Northern Raccoon (13)	97.90%	98.30%	0.981	98.50%	100.00%	0.992	99.20%	98.30%	0.987
Grey Fox (14)	87.10%	88.10%	0.876	85.80%	90.50%	0.881	87.10%	89.30%	0.882
Horse (15)	98.10%	91.70%	0.948	95.30%	91.70%	0.935	94.40%	91.70%	0.930
Dog (16)	97.20%	97.50%	0.973	96.50%	98.50%	0.975	96.00%	95.70%	0.958
American Crow (17)	77.00%	87.50%	0.819	72.60%	89.70%	0.802	73.70%	89.70%	0.809
Chicken (18)	81.00%	78.10%	0.795	79.20%	84.40%	0.817	77.80%	80.80%	0.793
Domestic Cat (19)	97.20%	97.90%	0.975	96.60%	97.90%	0.972	96.40%	99.00%	0.977
Coyote (20)	94.10%	97.00%	0.955	94.60%	98.50%	0.965	93.60%	98.50%	0.960
Bobcat (21)	93.60%	90.70%	0.921	92.30%	95.40%	0.938	87.20%	95.40%	0.911

Table A.6: Influence of input dimensionality reduction on the performance of Yolov5s model

		Y5sENA640xCropRGBTrain100			Y5sENA224xCropGWNTrain100			Y5sENA224xCropGWNTrain20			Y5sENA64xCropGNTrain100		
	Support	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	1757	90.24%	92.64%	0.913	89.99%	86.48%	0.881	78.69%	71.36%	0.745	61.53%	49.94%	0.540
All - Average	1757	90.50%	92.60%	0.915	89.70%	85.60%	0.876	77.50%	67.40%	0.721	64.70%	44.30%	0.526
Bird (0)	40	83.90%	91.50%	0.875	80.30%	72.50%	0.762	48.30%	37.40%	0.422	70.20%	25.00%	0.369
Eastern Gray Squirrel (1)	60	91.90%	94.40%	0.931	74.40%	63.00%	0.682	46.60%	33.30%	0.388	0.00%	0.00%	0
Eastern Chipmunk (2)	62	98.20%	90.30%	0.941	89.10%	67.70%	0.769	74.10%	46.20%	0.569	0.00%	0.00%	0
Woodchuck (3)	41	95.60%	97.60%	0.966	94.60%	95.10%	0.948	97.00%	77.90%	0.864	56.40%	37.90%	0.453
Wild Turkey (4)	58	73.00%	83.70%	0.780	75.50%	75.90%	0.757	62.70%	46.30%	0.533	42.70%	34.50%	0.382
White_Tailed_Deer (5)	68	86.70%	95.70%	0.910	89.40%	92.60%	0.910	87.40%	77.90%	0.824	80.20%	83.80%	0.820
Virginia Opossum (6)	145	99.50%	97.20%	0.983	99.30%	95.60%	0.974	97.80%	92.10%	0.949	80.10%	83.40%	0.817
Eastern Cottontail (7)	57	81.30%	93.00%	0.868	84.20%	87.70%	0.859	60.80%	64.90%	0.628	38.40%	5.49%	0.096
American Black Bear (8)	179	92.60%	93.30%	0.929	92.00%	89.70%	0.908	84.20%	86.60%	0.854	81.20%	76.50%	0.788
Vehicle (9)	28	83.20%	88.40%	0.857	79.50%	75.00%	0.772	88.00%	50.00%	0.638	90.70%	46.40%	0.614
Striped Skunk (10)	59	96.30%	89.20%	0.926	93.90%	89.80%	0.918	80.50%	76.80%	0.786	58.30%	40.20%	0.476
Red Fox (11)	83	98.70%	93.80%	0.962	94.00%	85.50%	0.895	82.00%	65.90%	0.731	52.20%	46.00%	0.489
Eastern Fox Squirrel (12)	68	93.00%	97.10%	0.950	94.00%	91.20%	0.926	86.30%	74.00%	0.797	58.10%	36.80%	0.451
Northern Raccoon (13)	58	99.50%	98.30%	0.989	98.20%	92.30%	0.952	80.40%	70.50%	0.751	58.60%	31.00%	0.405
Grey Fox (14)	84	85.10%	88.20%	0.866	93.60%	86.60%	0.900	81.50%	73.30%	0.772	51.30%	40.20%	0.451
Horse (15)	12	93.30%	91.70%	0.925	93.40%	91.70%	0.925	88.10%	83.30%	0.856	93.10%	75.00%	0.831
Dog (16)	140	96.10%	98.60%	0.973	97.30%	94.30%	0.958	90.10%	85.00%	0.875	76.50%	62.90%	0.690
American Crow (17)	184	74.10%	88.00%	0.805	76.80%	81.00%	0.788	69.10%	75.50%	0.722	59.50%	61.40%	0.604
Chicken (18)	104	80.50%	78.80%	0.796	81.80%	77.60%	0.796	63.50%	56.70%	0.599	60.70%	43.30%	0.505
Domestic Cat (19)	96	98.90%	97.50%	0.982	96.20%	91.70%	0.939	89.50%	71.90%	0.797	52.60%	37.50%	0.438
Coyote (20)	66	96.40%	98.50%	0.974	98.40%	95.40%	0.969	91.50%	82.10%	0.865	91.30%	79.10%	0.848
Bobcat (21)	65	92.60%	92.30%	0.924	96.70%	90.10%	0.933	56.00%	55.40%	0.557	70.60%	29.20%	0.413

Table A.7: Performance of VAE-SVM-4M versus VAE-SVM-665k on ENA64xCrop dataset

		VAE-SVM-4M			VAE-SVM-665K		
Class	support	precision	recall	f1-score	precision	recall	f1-score
All - Average	1757	66.38%	63.63%	0.632	67.38%	63.69%	**0.634**
All - Weighed	1757	64.62%	55.07%	0.579	65.36%	55.41%	**0.581**
Bird (0)	40	73.08%	47.50%	0.576	75.00%	52.50%	**0.618**
Eastern Gray Squirrel (1)	60	92.50%	61.67%	0.740	97.37%	61.67%	**0.755**
Eastern Chipmunk (2)	62	96.67%	93.55%	**0.951**	94.92%	90.32%	0.926
Woodchuck (3)	41	96.43%	65.85%	**0.783**	100.00%	60.98%	0.758
Wild Turkey (4)	58	84.00%	36.21%	**0.506**	80.95%	29.31%	0.430
White_Tailed_Deer (5)	68	59.72%	63.24%	**0.614**	47.00%	69.12%	0.560
Virginia Opossum (6)	145	86.06%	97.93%	0.916	86.59%	97.93%	**0.919**
Eastern Cottontail (7)	57	37.50%	15.79%	0.222	41.67%	17.54%	**0.247**
American Black Bear (8)	179	40.23%	77.09%	0.529	41.02%	76.54%	**0.534**
Vehicle (9)	28	55.00%	39.29%	0.458	48.15%	46.43%	**0.473**
Striped Skunk (10)	59	49.06%	44.07%	**0.464**	48.08%	42.37%	0.450
Red Fox (11)	83	39.36%	44.58%	0.418	44.19%	45.78%	**0.450**
Eastern Fox Squirrel (12)	68	81.03%	69.12%	**0.746**	73.77%	66.18%	0.698
Northern Raccoon (13)	58	80.65%	43.10%	**0.562**	82.76%	41.38%	0.552
Grey Fox (14)	84	59.49%	55.95%	0.577	76.19%	57.14%	**0.653**
Horse (15)	12	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Dog (16)	140	75.38%	70.00%	0.726	82.20%	69.29%	**0.752**
American Crow (17)	184	71.04%	85.33%	**0.775**	67.39%	84.24%	0.749
Chicken (18)	104	86.73%	81.73%	**0.842**	85.86%	81.73%	0.837
Domestic Cat (19)	96	51.95%	41.67%	0.462	54.79%	41.67%	**0.473**
Coyote (20)	66	76.32%	43.94%	0.558	82.93%	51.52%	**0.636**
Bobcat (21)	65	29.33%	33.85%	**0.314**	27.06%	35.38%	0.307

A.2 Class-wise Islands dataset results

This appendix section contains class-wise results from the experiments run on the Channel Islands [18] dataset. Reported scores are results from classification on the test set.

Table A.8: Comparison: Best performing Ghost model and Yolov5s models on ISL224xCropTrain5 and ISL224xSeqTrain5 data

		Ghostnet: Seq.			Yolov5s: Crop.			Yolov5s: Seq.		
Class	support	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
All - Weighed	8593	74.37%	72.98%	0.732	43.69%	48.62%	0.432	81.43%	81.88%	**0.810**
Empty (0)	3981	77.32%	82.65%	0.799	62.90%	91.80%	0.747	79.60%	88.70%	**0.839**
Other (1)	2	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Fox (2)	1831	60.29%	68.83%	0.643	68.30%	28.60%	0.403	94.20%	67.20%	**0.784**
Skunk (3)	24	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Rodent (4)	2544	86.83%	67.64%	0.760	0.00%	0.00%	0.000	80.30%	87.70%	**0.838**
Bird (5)	212	0.00%	0.00%	0.000	0.00%	0.00%	0.000	28.80%	20.30%	**0.238**

Table A.9: Performance of VAE-SVM-4M versus VAE-SVM-665k on ISL64xCrop dataset

		VAE-SVM-4M			VAE-SVM-665k		
Class	support	precision	recall	f1-score	precision	recall	f1-score
All - Weighed	8593	78.64%	79.91%	**0.789**	77.42%	79.01%	0.779
Empty (0)	3981	82.97%	82.84%	**0.829**	82.19%	83.35%	0.828
Other (1)	2	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Fox (2)	1831	74.19%	72.69%	**0.734**	72.46%	68.10%	0.702
Skunk (3)	24	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Rodent (4)	2544	79.61%	87.66%	**0.834**	78.86%	87.22%	0.828
Bird (5)	212	33.33%	3.77%	**0.068**	22.73%	2.36%	0.043

Table A.10: Performance of VAE-4M and Yolov5s on ISL64x Crop and Seq. datasets

		VAE-SVM-4M: crop data			VAE-SVM-4M: seq. data			Yolov5s: crop data			Yolov5s: seq. data		
Class	support*	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score
All - Weighed	8593	78.64%	79.91%	**0.789**	71.04%	71.58%	0.702	37.01%	44.61%	0.358	44.79%	62.27%	0.496
Empty (0)	3981	82.97%	82.84%	**0.829**	84.79%	78.15%	0.813	45.90%	84.10%	0.594	37.20%	76.10%	0.500
Other (1)	2	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Fox (2)	1831	74.19%	72.69%	**0.734**	64.56%	44.67%	0.528	73.90%	26.50%	0.390	66.50%	43.70%	0.527
Skunk (3)	24	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000
Rodent (4)	2544	79.61%	87.66%	**0.834**	60.79%	87.34%	0.717	0.00%	0.00%	0.000	45.20%	59.80%	0.515
Bird (5)	212	33.33%	3.77%	**0.068**	0.00%	0.00%	0.000	0.00%	0.00%	0.000	0.00%	0.00%	0.000

*Yolov5s tested on one (1) fewer 'empty' class image. This was taken into account for calculations.

Appendix B

Appendix: miscellaneous

This appendix chapter contains the appendixes on the number of parameters in the VAE models, class distributions, and information regarding the test and validation subsets.

B.1 VAE: number of trainable parameters

This section showcases the number of trainable parameters of the VAE models. This does not include the SVM support vector parameters (weights) which are part of the VAE-SVM model.

Model parameters of VAE 665k model:

Modules	Parameters
encoder.encoder.0.0.weight	864
encoder.encoder.0.0.bias	32
encoder.encoder.0.1.weight	32
encoder.encoder.0.1.bias	32
encoder.encoder.1.0.weight	18432
encoder.encoder.1.0.bias	64
encoder.encoder.1.1.weight	64
encoder.encoder.1.1.bias	64
encoder.encoder.2.0.weight	36864
encoder.encoder.2.0.bias	64
encoder.encoder.2.1.weight	64
encoder.encoder.2.1.bias	64
encoder.encoder.3.0.weight	73728
encoder.encoder.3.0.bias	128
encoder.encoder.3.1.weight	128
encoder.encoder.3.1.bias	128
encoder.encoder.4.0.weight	147456
encoder.encoder.4.0.bias	128
encoder.encoder.4.1.weight	128
encoder.encoder.4.1.bias	128
encoder.fc_mu.weight	32768
encoder.fc_mu.bias	64
encoder.fc_var.weight	32768
encoder.fc_var.bias	64
decoder_input.weight	32768
decoder_input.bias	512
decoder.0.0.weight	147456
decoder.0.0.bias	128
decoder.0.1.weight	128
decoder.0.1.bias	128
decoder.1.0.weight	73728
decoder.1.0.bias	64
decoder.1.1.weight	64

decoder.1.1.bias	64
decoder.2.0.weight	36864
decoder.2.0.bias	64
decoder.2.1.weight	64
decoder.2.1.bias	64
decoder.3.0.weight	18432
decoder.3.0.bias	32
decoder.3.1.weight	32
decoder.3.1.bias	32
final_layer.0.weight	9216
final_layer.0.bias	32
final_layer.1.weight	32
final_layer.1.bias	32
final_layer.3.weight	864
final_layer.3.bias	3

Total Trainable Params full VAE model: 665059

Model parameters of VAE 665k model, encoder only:

Modules	Parameters
encoder.encoder.0.0.weight	864
encoder.encoder.0.0.bias	32
encoder.encoder.0.1.weight	32
encoder.encoder.0.1.bias	32
encoder.encoder.1.0.weight	18432
encoder.encoder.1.0.bias	64
encoder.encoder.1.1.weight	64
encoder.encoder.1.1.bias	64
encoder.encoder.2.0.weight	36864
encoder.encoder.2.0.bias	64
encoder.encoder.2.1.weight	64
encoder.encoder.2.1.bias	64
encoder.encoder.3.0.weight	73728
encoder.encoder.3.0.bias	128
encoder.encoder.3.1.weight	128
encoder.encoder.3.1.bias	128
encoder.encoder.4.0.weight	147456
encoder.encoder.4.0.bias	128
encoder.encoder.4.1.weight	128
encoder.encoder.4.1.bias	128
encoder.fc_mu.weight	32768
encoder.fc_mu.bias	64
encoder.fc_var.weight	32768
encoder.fc_var.bias	64

Total Trainable Params Encoder: 344256

Model parameters of VAE-4M model:

Modules	Parameters
encoder.encoder.0.0.weight	864
encoder.encoder.0.0.bias	32
encoder.encoder.0.1.weight	32
encoder.encoder.0.1.bias	32
encoder.encoder.1.0.weight	18432
encoder.encoder.1.0.bias	64
encoder.encoder.1.1.weight	64
encoder.encoder.1.1.bias	64
encoder.encoder.2.0.weight	73728
encoder.encoder.2.0.bias	128
encoder.encoder.2.1.weight	128
encoder.encoder.2.1.bias	128
encoder.encoder.3.0.weight	294912
encoder.encoder.3.0.bias	256
encoder.encoder.3.1.weight	256
encoder.encoder.3.1.bias	256
encoder.encoder.4.0.weight	1179648
encoder.encoder.4.0.bias	512
encoder.encoder.4.1.weight	512
encoder.encoder.4.1.bias	512
encoder.fc_mu.weight	262144
encoder.fc_mu.bias	128
encoder.fc_var.weight	262144
encoder.fc_var.bias	128
decoder_input.weight	262144
decoder_input.bias	2048
decoder.0.0.weight	1179648
decoder.0.0.bias	256
decoder.0.1.weight	256
decoder.0.1.bias	256
decoder.1.0.weight	294912
decoder.1.0.bias	128
decoder.1.1.weight	128
decoder.1.1.bias	128
decoder.2.0.weight	73728
decoder.2.0.bias	64
decoder.2.1.weight	64
decoder.2.1.bias	64
decoder.3.0.weight	18432
decoder.3.0.bias	32
decoder.3.1.weight	32
decoder.3.1.bias	32
final_layer.0.weight	9216
final_layer.0.bias	32
final_layer.1.weight	32
final_layer.1.bias	32
final_layer.3.weight	864
final_layer.3.bias	3

Total Trainable Params: 3937635

Model parameters of VAE 665k model, encoder only:

Modules	Parameters
encoder.encoder.0.0.weight	864
encoder.encoder.0.0.bias	32
encoder.encoder.0.1.weight	32
encoder.encoder.0.1.bias	32
encoder.encoder.1.0.weight	18432
encoder.encoder.1.0.bias	64
encoder.encoder.1.1.weight	64
encoder.encoder.1.1.bias	64
encoder.encoder.2.0.weight	73728
encoder.encoder.2.0.bias	128
encoder.encoder.2.1.weight	128
encoder.encoder.2.1.bias	128
encoder.encoder.3.0.weight	294912
encoder.encoder.3.0.bias	256
encoder.encoder.3.1.weight	256
encoder.encoder.3.1.bias	256
encoder.encoder.4.0.weight	1179648
encoder.encoder.4.0.bias	512
encoder.encoder.4.1.weight	512
encoder.encoder.4.1.bias	512
encoder.fc_mu.weight	262144
encoder.fc_mu.bias	128
encoder.fc_var.weight	262144
encoder.fc_var.bias	128

Total Trainable Params Encoder: 2095104

B.2 Subset class distributions

This section showcases the distributions of the classes in both the ENA24 [61] and Channel Islands [18] subsets.

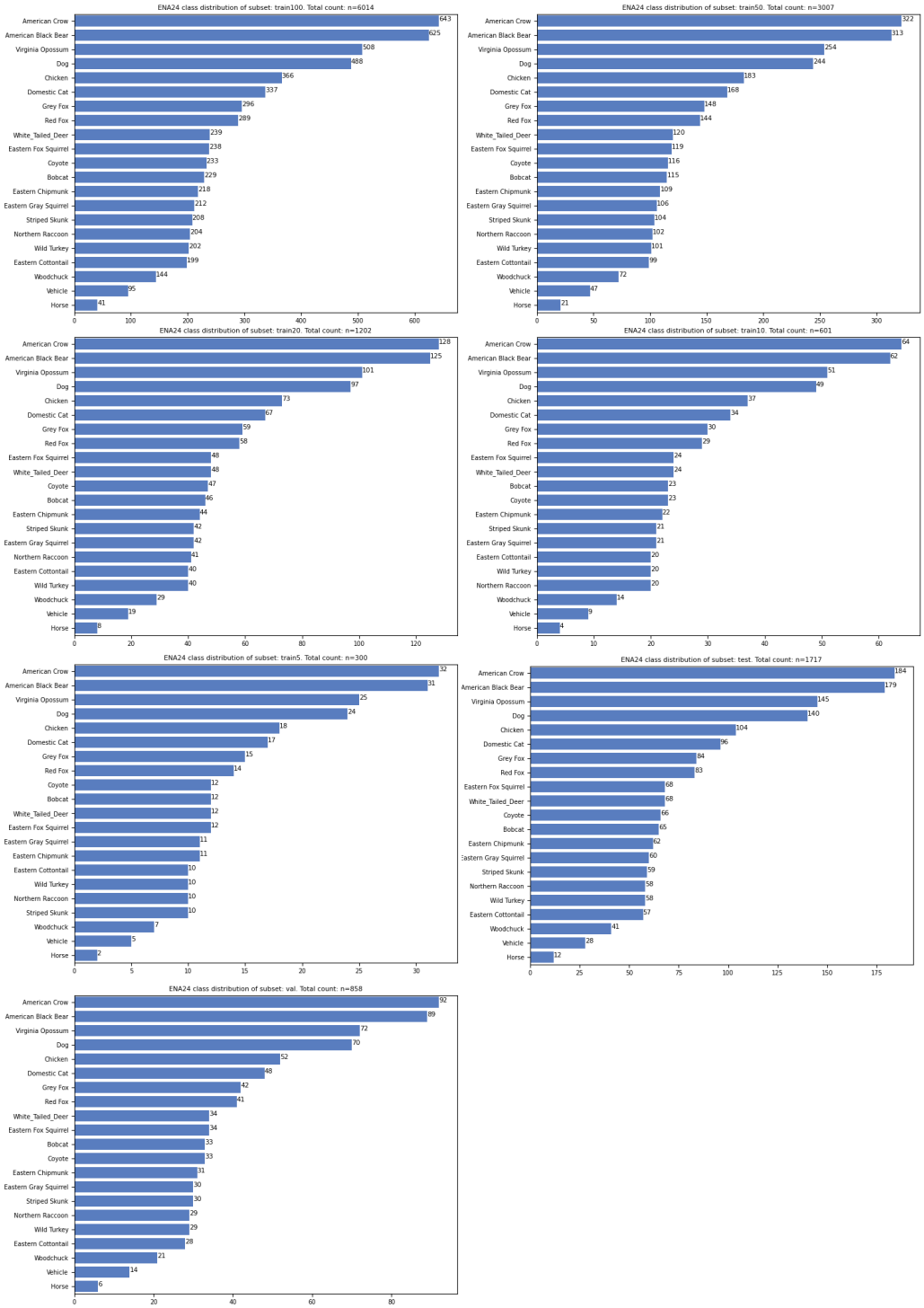


Figure B.1: Class distributions of ENA subsets.

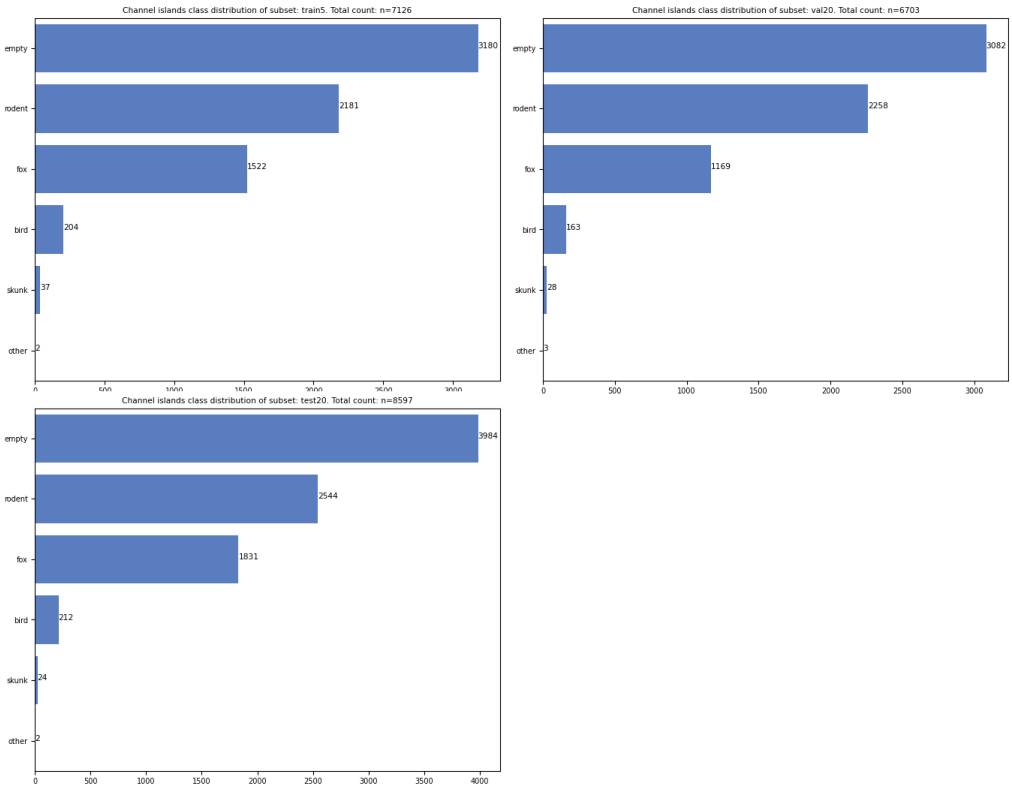


Figure B.2: Class distributions of Channel Islands subsets.

B.3 Test and validation subsets

This section contains the test and validation subsets for both the ENA24 [61] and Channel Islands [18] datasets. The naming of the subsets indicate which steps were applied to the images during the processing in the data pipeline. The set of images used for test and validation were selected prior to model training once, and not changed.

Table B.1: Channel Islands dataset [18] test and validation subsets

Subset name	Subset size	Image dimensions	Images	Avg. size (kb)
ISLORIGxCropRGBTest20	5.5 GB	original	8594	639.981
ISL224xCropRGBTest20	174.7 MB	224x224	8594	20.328
ISL224xSeqRGBTest20	159.4 MB	224x224	8594	18.548
ISL224xCropRGBVal20	136.3 MB	224x224	6703	20.334
ISL224xSeqRGBVal20	122.8 MB	224x224	6703	18.320
ISL64xCropRGBTest20	22.9 MB	64x64	8594	2.665
ISL64xSeqRGBTest20	21.9 MB	64x64	8594	2.548
ISL64xCropRGBVal20	17.8 MB	64x64	6703	2.656
ISL64xSeqRGBVal20	16.9 MB	64x64	6703	2.521

Table B.2: ENA24 dataset [61] test and validation subsets

Subset name	Subset size	Image dimensions	Images	Avg. size (kb)
ENAORIGxCropRGBTest	1.4 GB	original	1757	796.813
ENAORIGxCropRGBVal	699.1 MB	original	878	796.241
ENA640xCropRGBTest	260.9 MB	640x640	1757	148.492
ENA640xCropRGBVal	130.0 MB	640x640	878	148.064
ENA640xCropGTest	238.8 MB	640x640	1757	135.913
ENA640xCropGVal	118.9 MB	640x640	878	135.421
ENA640xCropGWTest	211.4 MB	640x640	1757	120.319
ENA640xCropGWVal	105.3 MB	640x640	878	119.932
ENA640xCropGNTest	104.2 MB	640x640	1757	59.306
ENA640xCropGNVal	51.9 MB	640x640	878	59.112
ENA640xCropGWNTest	90.4 MB	640x640	1757	51.451
ENA640xCropGWNVal	45.1 MB	640x640	878	51.367
ENA224xCropGWNTest	15.3 MB	224x224	1757	8.708
ENA224xCropGWNVal	7.7 MB	224x224	878	8.770
ENA64xCropGNTest	2.3 MB	64x64	1757	1.309
ENA64xCropGNVal	1.2 MB	64x64	878	1.367