



B.Sc. (Hons) in Software Development



Ollscoil
Teicneolaíochta
an Atlantaigh

Atlantic
Technological
University

CritterSnap: Automated Event-Based Wildlife Image Classification Using Edge Detection and Machine Learning

By
Ronan Francis

April 28, 2025

Minor Dissertation

**Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.**

Contents

1	Introduction	7
1.1	Project Context	7
1.2	Project Objectives	8
1.2.1	Success Metrics	9
1.3	Technologies and System Architecture	10
1.3.1	Technologies Used	10
1.3.2	System Architecture	10
1.4	Dissertation Structure	11
2	Methodology	12
2.1	Development Model	12
2.2	Software Development Process	13
2.2.1	Development Approach: Iterative and Modular Design . . .	13
2.2.2	Implementation Phases	13
2.3	Testing and Validation	15
2.4	Technologies and Tools	15
3	Technology Review	16
3.1	Software Libraries and Computational Framework	16
3.1.1	Built-in Python Modules (os, json, datetime, shutil)	16
3.1.2	Numerical and Array Operations (NumPy)	17
3.1.3	Parallel and Distributed Processing (concurrent.futures and joblib)	17
3.2	Image Processing and Detection Algorithms	17
3.2.1	Edge Detection and Histogram Features	17
3.2.2	Dynamic Thresholding and Binarization	19
3.2.3	Motion Detection via Structural Similarity Index (SSIM) .	20
3.3	Machine Learning: One-Class Classification	21
3.4	GDPR Filtering and Ethical Image Processing	22
3.5	Tools Considered but Rejected	23
3.5.1	Deep Learning Classifiers	23

3.5.2	Cloud-Based APIs (Google Cloud Vision)	23
4	System Design	25
4.1	Working with Images	26
4.1.1	Data Input and Preprocessing	26
4.1.2	GDPR Mask Detection	26
4.1.3	Threaded I/O and Preprocessing	27
4.2	Sequence Reconstruction	27
4.2.1	Event-Based Image Grouping	27
4.2.2	Sequence Reconstruction Algorithm	28
4.2.3	Effectiveness of Sequence Reconstruction	29
4.3	Motion and Change Detection	29
4.3.1	Composite Scoring Approach	29
4.3.2	Sobel Edge Detection	31
4.4	Best Photo Selection and Classification	31
4.4.1	Best Representative Image	31
4.4.2	One-Class SVM Classification	32
4.5	Complete System Pipeline	32
4.5.1	Output Organization	34
5	System Evaluation	35
5.1	Evaluation of System Objectives	35
5.2	System Robustness and Stability	36
5.2.1	Unit Testing	36
5.2.2	Acceptance Testing	36
5.2.3	Structural Stability	36
5.3	Performance and Accuracy	36
5.3.1	Speed and Scalability	37
5.3.2	Classification Accuracy	37
5.4	Limitations and Opportunities	37
5.4.1	Future Improvements	38
6	Conclusion	39
A	Repository and Installation Instructions	42
A.1	Repository Information	42
A.2	Installation Instructions	42
A.2.1	Prerequisites	42
A.2.2	Step 1: Clone the Repository	43
A.2.3	Step 2: Set Up a Virtual Environment (Recommended)	43
A.2.4	Step 3: Install Dependencies	43

A.2.5	Step 4: Run the Application	43
A.2.6	Troubleshooting	44

List of Figures

1.1	“Layer cake” deployment stack	10
2.1	System Architecture of CritterSnap	12
2.2	Two-Step Classification Process in CritterSnap	14
2.3	Ghannt chart timeline for project	15
3.1	Comparison of edge detection algorithms (Sobel, Prewitt, Canny) across multiple grayscale images. Sobel provides strong directional gradients ideal for histogram-based feature extraction, while Canny offers finer details at a higher computational cost.Prewitt offers a lightweight but less precise alternative.	18
3.2	Example of SSIM-based comparison between two similar frames. The SSIM score of 0.91 indicates high perceptual similarity. The rightmost image visualizes areas of structural difference, used by CritterSnap to detect motion without relying on raw pixel deltas. .	20
4.1	Overall system architecture of CritterSnap.	25
4.2	Horizontal 60-second grouping timeline	28
A.1	(a) GitHub repository page and (b) CritterSnap demo output. . . .	44

List of Tables

3.1	Technology Selection Criteria for CritterSnap System	24
5.1	Performance Evaluation	37
5.2	Classification Results on Test Dataset	37

Field cameras generate images at a pace that exceeds our capacity to tag them efficiently, and transferring these images to the cloud can pose practical and legal challenges, especially concerning GDPR. CritterSnap has been created as a user-friendly computational pipeline for laptops to tackle this issue. It efficiently organizes wildlife images, accurately identifies authentic wildlife activity, and permits users to integrate their labeled datasets, facilitating model retraining based on environmental changes and data evolution.

The workflow involves several steps: first, it stitches photos into bursts by time and location; next, it evaluates each burst to determine the clearest “story” frame using Sobel edge density and SSIM measures; finally, these frames are processed using a support vector machine (SVM). Users can choose between a preconfigured one-class model or train a binary model with their datasets; all managed via a configuration file. A quick check for white pixels ensures compliance with GDPR by filtering out masked images.

When tested on a mixed woodland dataset of 5,000 images, the model achieved 91% accuracy, 94% recall, and a low 4.6% false positive rate. It processed 1,000 photos in just 2 minutes and 43 seconds, greatly exceeding project speed and precision goals. All resources are open-sourced, allowing conservation teams to adapt CritterSnap to their needs.

Chapter 1

Introduction

This research addresses a critical challenge in modern wildlife conservation: efficiently processing and accurately categorizing the vast amounts of image data collected by camera traps in the field. Conservationists often gather thousands of images that require manual inspection—a time-consuming and error-prone process, especially when the images are pre-processed to anonymize human faces for GDPR compliance.

The system described here automatically reconstructs photo sequences and labels photographs as "events" (instances when animals appear) or "non-events." It accomplishes this through metadata analysis, motion detection, and a specially developed machine-learning algorithm designed for one-class classification.

The project targets an audience with a moderately technical background and does not require prior expertise in wildlife monitoring or machine learning. Each component is explained in basic terms, allowing conservationists, data scientists, and tech practitioners to understand the problem and the proposed solution.

1.1 Project Context

Wildlife conservationists are increasingly turning to technology to monitor and protect animal populations. Camera traps have become a standard tool[1] for collecting photographic evidence of wildlife across diverse ecosystems. However, the sheer volume of data generated- ranging from 1,200 to as many as 20,000 photos in some cases - can be overwhelming for human analysis. Current image classification tools like EcoAssist can produce errors under certain conditions. One specific issue is the mislabeling of images containing GDPR-protected information. For example, when images are pre-processed to mask human faces with white rectangles (a precaution taken to comply with GDPR), these images may be incorrectly labeled as featuring animal occurrences.

In most conservation studies, such as the Snapshot Europe Study, the procedure involves gathering a set of sequential photographs into a single "sequence" for labeling rather than labeling each photograph individually. Typically a camera trap can capture images within a 60-second window when it detects movement. These images are then stitched together into a sequence. For instance, a sequence could consist of nine images showing different phases of animal movement. In one case, the first three might depict a single deer, the following three could show two deer, and the final three might illustrate the same deer again. This entire sequence would then be recorded as "2 deer", counted as one entry within the dataset.

This project will result in an automated system that reconstructs these sequences based on metadata such as time and location. Since no existing metadata directly links individual photographs to their sequences, the system groups photos with similar timestamps and location data using a 60-second bundling window.. This approach should effectively reconstruct sequences for observed species, ensuring that conservationists have access to accurate data regarding wildlife activities.

The system will use motion capture to detect changes between frames and identify potential animal presence. It will utilize machine learning techniques to classify images into events (those with animals) and non-events (those without animals), as well as classify specific species (e.g., deer, bat, sheep). The use of a machine learning-based solution is critical in addressing the issue of misclassification caused by white-boxed images, improving upon the limitations of current tools like EcoAssist.[2]

This research also has broader applications beyond image processing alone. Automating the classification of wildlife camera traps and images will free conservationists from handling data processing and channel more resources toward protection efforts. Further, the high-speed capability of handling large data sets enables more comprehensive tracking of vulnerable species populations and their patterns, which could lead to more effective conservation strategies.[3]

1.2 Project Objectives

The primary aims of this project are as follows:

- **Sequence Reconstruction:** A method will be developed to reassemble image sequences from individual photos using time and location metadata. By grouping photos captured within 60 seconds and from the same location, sequences that reflect real-world animal movement events can be reconstructed.
- **Motion Detection Utilization:** Motion detection techniques will be employed to detect significant changes between sequential frames. This ap-

proach assists in identifying potential animal presence while reducing false positives caused by minor environmental changes or noise.

- **One-Class Classification Model:** A machine learning workflow will be implemented to classify events using a one-class classifier. Given the exclusive availability of event sequences (i.e., sequences where animals are present), this model will enable differentiation between events and non-events without requiring non-event training examples.
- **GDPR Compliance:** Handling of images pre-processed to obscure human faces (e.g., through white-box overlays) will be ensured. This measure is essential for preventing misclassification and excluding privacy-sensitive images from animal event classifications.
- **Data Pipeline Development:** An automated data pipeline will be constructed to support the complete workflow—from image acquisition and pre-processing to sequence reconstruction, classification, and output generation. The pipeline will be designed for ease of use by conservationists supplying folders of GDPR-compliant images.
- **Performance Evaluation:** Model performance will be evaluated using cross-validation techniques and metrics such as accuracy, precision, recall, and F1-score. These evaluations will verify the classifier's suitability for real-world applications.
- **Local Deployment:** A standalone program will be developed for execution on local machines, enabling use in field environments without relying on cloud services or complex infrastructure.

1.2.1 Success Metrics

The success of this project will be quantified by the use of the following metrics:

- **Classification Accuracy:** As a proportion of images correctly classified (target: $> 90\%$)
- **Processing Speed:** Time to process 1,000 images (target: < 5 minutes on standard hardware)
- **False Positive Rate:** As a proportion of non-animal images incorrectly classified as containing animals (target: $< 5\%$)
- **Sequence Reconstruction Accuracy:** Percentage of correctly grouped image sequences (target: $> 95\%$)

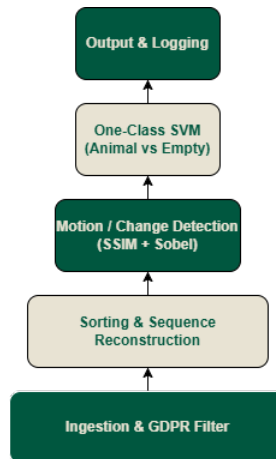


Figure 1.1: “Layer cake” deployment stack

1.3 Technologies and System Architecture

1.3.1 Technologies Used

- **Python:** Main programming language.
- **Image Processing:** Pillow (PIL), NumPy, scikit-image.
- **Machine Learning:** scikit-learn (OneClassSVM).
- **Concurrency:** `concurrent.futures` (ProcessPoolExecutor, ThreadPoolExecutor).
- **Utilities:** `os`, `datetime`, `shutil`.

1.3.2 System Architecture

1. **Data Ingestion and Preprocessing:** Load images, filter GDPR-masked images, and extract timestamps.
2. **Image Sorting and Grouping:** Sort images by capture time and group them based on time gaps.
3. **First-Pass Event Detection:** Compute a composite score using SSIM and edge detection to classify events.
4. **Second-Pass AI-Based Classification:** Train a OneClassSVM on animal images and reclassify ambiguous cases.

5. **Output and Logging:** Log results and organize classified images into output directories.

1.4 Dissertation Structure

The remainder of this dissertation is structured as follows:

- **Chapter 2 – Methodology:** Describes the iterative development process and tools used.
- **Chapter 3 – Technology Review:** Evaluates prior tools, methods, and theoretical foundations.
- **Chapter 4 – System Design:** Details the architecture and structure of the CritterSnap system.
- **Chapter 5 – Evaluation:** Provides performance metrics and robustness testing.
- **Chapter 6 – Conclusion:** Summarizes findings, reflects on limitations, and proposes future work.

Chapter 2

Methodology

This chapter outlines the systematic approach used in developing and testing the CritterSnap system. The methodology fuses a modular software engineering framework with a research paradigm focused on image processing and binary classification. Development was conducted in iterative sprints, which allowed continuous updates based on experimental results and system requirements.

Figures 2.1 and 2.2 illustrate the complete pipeline, from initial acquisition to final classification.

2.1 Development Model

An Agile-inspired development model guided the implementation. The project was broken into short sprints, each focused on a specific feature or challenge (e.g., sequence reconstruction, motion analysis, classification tuning). This approach allowed for continuous integration and testing of new components and rapid re-

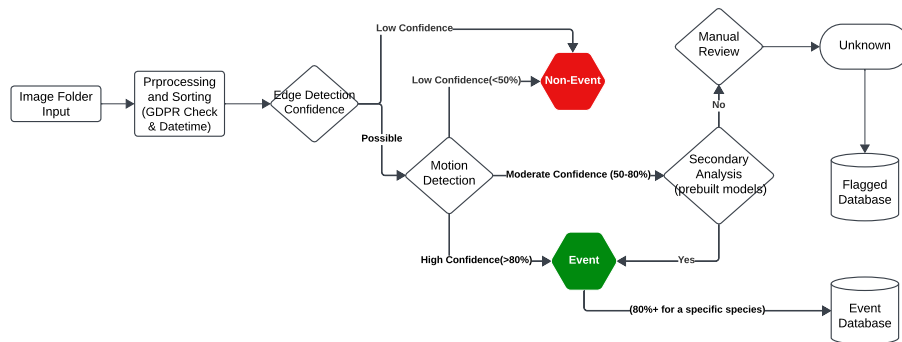


Figure 2.1: System Architecture of CritterSnap

sponses to feedback or observed issues. Development progress was reviewed weekly with the project supervisor.

2.2 Software Development Process

2.2.1 Development Approach: Iterative and Modular Design

The system was developed using independent, testable modules to promote clarity, re-usability, and performance. Key components—event detector, classifier, and reconstructor—were designed to operate as plug-and-play tools, allowing for rapid refinement of one part without destabilizing the whole pipeline.

- **Data Sorting:** Algorithms were built to extract and organize timestamps, even when metadata was missing.
- **Event Detection:** Frame differencing and GDPR marker checks were implemented as preprocessing layers.
- **Threaded Processing:** Image handling was multithreaded to minimize I/O bottlenecks and improve responsiveness when operating over large datasets.

2.2.2 Implementation Phases

1. **Setup and Research:** Literature was reviewed, and project goals were defined, including the identification of suitable machine learning algorithms and privacy constraints.
2. **Data Collection and Preprocessing:** A dataset of 5,000 images was sourced from the Frankfurt Conservation Society. Metadata was normalized, anonymized, and grouped for initial tests.
3. **Prototype Classifier:** A Decision Tree was initially implemented for basic classification tasks and later replaced with motion-based heuristics.
4. **One-Class Model Development:** Scikit-learn's OneClassSVM was adopted to distinguish event images using only positive examples.
5. **Optimization and Multithreading:** As dataset size increased, the system was refactored to handle image sets in parallel using Python's `concurrent.futures`.

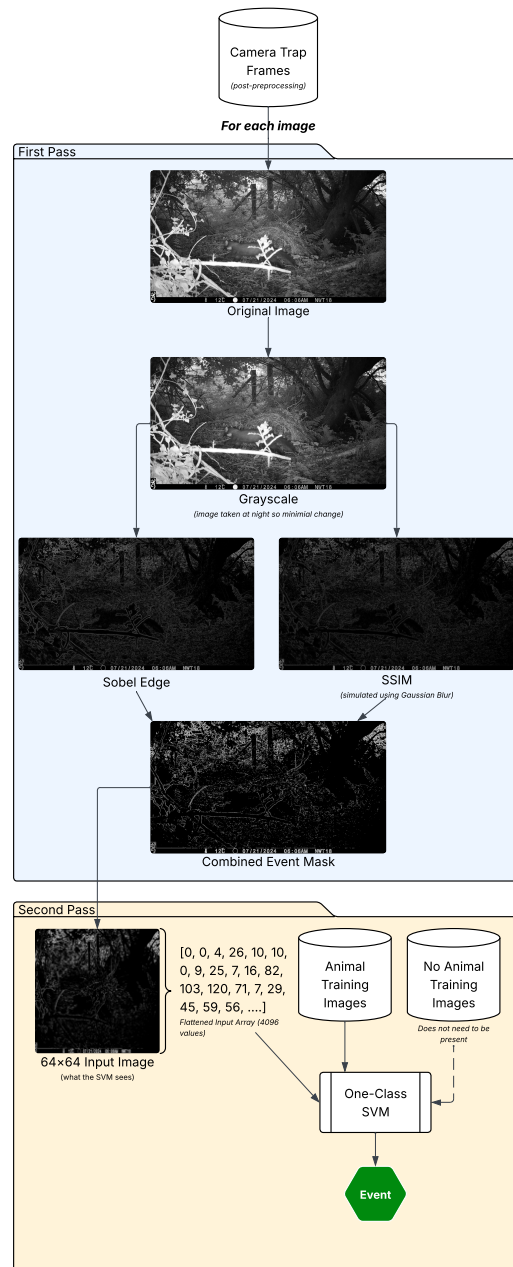


Figure 2.2: Two-Step Classification Process in CritterSnap

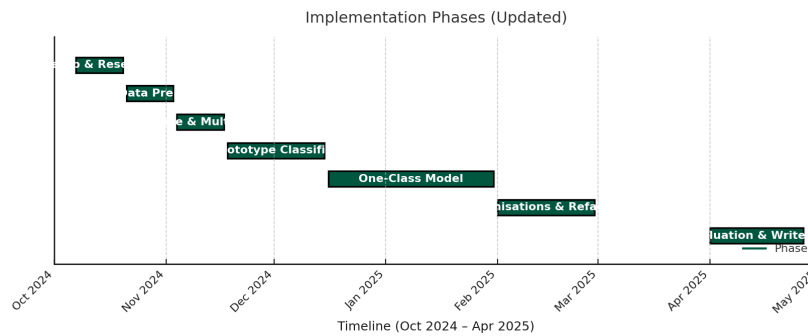


Figure 2.3: Ghannt chart timeline for project

2.3 Testing and Validation

Multiple forms of testing were applied:

- **Unit Testing:** Image parsing, metadata grouping, and GDPR filters were tested with small, known datasets.
- **Empirical Testing:** Larger image sets with manually labeled events were used to benchmark motion detection and classifier accuracy.
- **Thread Safety Testing:** Multithreaded pipeline components were stress-tested by duplicating input image folders to simulate extreme load.

Classifier performance was evaluated using cross-validation where applicable. The final metrics reported include accuracy, recall, and false positive rate.

2.4 Technologies and Tools

- **Language:** Python 3.11
- **Libraries:** Pillow, scikit-image, scikit-learn, NumPy
- **Development Tools:** Visual Studio Code, GitHub, Markdown for logs
- **Concurrency:** `concurrent.futures` for parallel processing
- **Testing:** Custom test scripts and manual inspection of output logs

Chapter 3

Technology Review

This chapter provides a literature-based analysis of the core technologies used in the development of CritterSnap. Sections 3.1 through 3.5 examine the system from low-level libraries to high-level classification, followed by a summary of rejected alternatives and a design rationale table. Each tool, algorithm and design choice is evaluated not only for its technical functionality but also for how it supports the constraints of the system in the real world: minimal supervision, ethical data handling, and efficient offline processing. The goal is to demonstrate that the chosen technologies are not only functional, but also defensible and well-reasoned when compared against competing alternatives. The technologies reviewed in this chapter underpin the system architecture described in Chapter 4 and are evaluated in practice in Chapter 5 against real-world datasets.

3.1 Software Libraries and Computational Framework

The system combines several essential libraries for tasks such as image pre-processing, feature extraction, classification, and GDPR compliance.

3.1.1 Built-in Python Modules (`os`, `json`, `datetime`, `shutil`)

The project makes extensive use of built-in Python modules, including `os`, `json`, `datetime`, and `shutil`, to handle file system operations, configuration management, and time-related data [4, 5, 6, 7, 8]. The `os` module provides functions for recursive directory traversal and file manipulation, which are essential for loading large image datasets. The `json` module enables configuration file serialization, allowing dynamic parameter adjustment. The `datetime` module is used to parse

and process time-based metadata from image files, which is vital for sequence reconstruction. The `shutil` module optimizes batch file operations, such as copying grouped image sequences into output directories.

3.1.2 Numerical and Array Operations (NumPy)

NumPy serves as the core library for numerical computations in the project. It converts image data into arrays and performs mathematical operations such as flattening and matrix multiplication. For example, image pre-processing functions flatten images into arrays to enable manipulation and analysis. Vectorized operations in NumPy reduce computational overhead significantly, essential while processing thousands of images in parallel. This efficiency enables the project to scale its image processing pipeline and ensures that the one-class classifier is exposed to normalized, consistent data structures [9].

3.1.3 Parallel and Distributed Processing (`concurrent.futures` and `joblib`)

The project uses parallel processing techniques to handle the complex computations involved in image processing and machine learning tasks. Specifically, Python's `concurrent.futures` module is employed to run tasks such as image sorting and GDPR filtering in thread and process pools [10]. This approach enables the system to process multiple folders of images concurrently and improves throughput on multi-core systems. Additionally, `joblib` is used for parallel execution of machine learning tasks, particularly when validating or testing different classifier configurations [11].

3.2 Image Processing and Detection Algorithms

3.2.1 Edge Detection and Histogram Features

A key feature of the CritterSnap system is its use of edge detection to extract image features for classification. Sobel filters compute horizontal and vertical gradients, which are then aggregated into directional histograms. These histograms are used as inputs to the machine-learning model. This approach is inspired by classical vision pipelines, where edge information is considered robust to lighting conditions and background noise [12]. Unlike deep neural network embeddings, edge-based features are interpretable, lightweight, and fast to compute — ideal for resource-constrained field deployment.



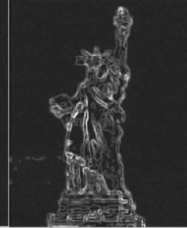
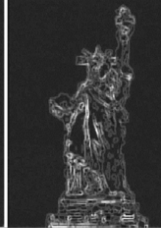

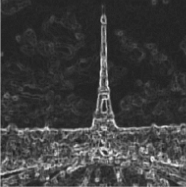
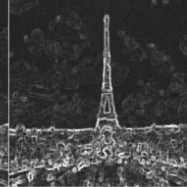
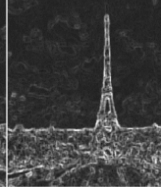


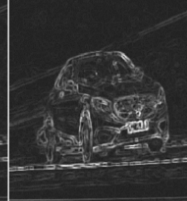
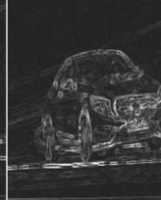




Image name	Gray scale image	Sobel operator	Prewitt operator	Canny operator
Statue of liberty				
Evil Tower				
Car				
Japanese House				

Figure 3.1: Comparison of edge detection algorithms (Sobel, Prewitt, Canny) across multiple grayscale images. Sobel provides strong directional gradients ideal for histogram-based feature extraction, while Canny offers finer details at a higher computational cost. Prewitt offers a lightweight but less precise alternative.

The Canny edge detector, known for its accuracy in identifying object boundaries, was ultimately excluded due to its computational cost and sensitivity to noise in natural environments. Canny’s multi-stage pipeline — involving Gaussian smoothing, gradient calculation, non-maximum suppression, and double thresholding — increases both processing time and implementation complexity. Given that CritterSnap operates on thousands of frames per session, the additional runtime was unjustified.

Prewitt filters, another gradient-based method similar to Sobel, were tested but produced weaker contrast in low-light forest images. Laplacian-based operators, while powerful in detecting rapid intensity changes, were overly sensitive to minor fluctuations such as background foliage movement or lighting variations.

Furthermore, although convolutional neural networks (CNNs) can learn highly abstract edge-like features automatically during training, they were considered

impractical here. CNNs require significantly more computational resources, annotated training data, and opaque post-hoc interpretability methods, making them less suited for an explainable, local deployment system like CritterSnap.

- **Sobel:** Fast, simple, robust — chosen
- **Canny:** Too slow, sensitive to noise
- **Prewitt:** Weak gradients in low-light
- **Laplacian:** Too sensitive to foliage flicker
- **CNNs:** Incompatible with explainability + deployment goals

Ultimately, Sobel filtering was chosen due to its low computational cost, effectiveness in varying lighting conditions, and compatibility with parallel processing. These benefits make it highly efficient for high-throughput wildlife image processing, where both interpretability and speed are crucial.

3.2.2 Dynamic Thresholding and Binarization

CritterSnap employs dynamic thresholding in its classification pipeline to improve robustness under varying environmental conditions. Instead of relying on fixed thresholds applied uniformly across datasets, the system derives context-sensitive values for edge confidence and motion change metrics from a small sample of known animal training data. This allows the system to calibrate itself to local lighting conditions, background noise, and species morphology — factors that often cause instability in static configurations.

The dynamic thresholding mechanism works by sampling a user-defined proportion of the available animal images (default: 10%) and estimating mean edge confidence scores and change measures using image triplets (i.e., sequential frames taken from the same burst or time window). These computed values are then used to update the `edge_confidence_threshold` and `change_threshold` parameters at runtime. Threshold calibration is performed once per classification session to ensure the system remains responsive to the dataset’s specific characteristics without compromising computational efficiency.

This approach outperforms fixed global thresholds, which often fail in forested environments where shadows, foliage, and ambient light vary significantly. Traditional pixel-level adaptive thresholding methods — such as Gaussian-weighted local thresholding — were initially tested but proved too computationally expensive and unreliable in low-contrast or backlit scenarios. In contrast, the implemented dynamic method updates thresholds globally but contextually, striking a practical balance between adaptability and performance.



Figure 3.2: Example of SSIM-based comparison between two similar frames. The SSIM score of 0.91 indicates high perceptual similarity. The rightmost image visualizes areas of structural difference, used by CritterSnap to detect motion without relying on raw pixel deltas.

Dynamic calibration also addresses one of the key challenges in outdoor image processing: the extreme variability in camera trap footage caused by environmental diversity across time and space. Prior research supports the use of statistical sampling to tune thresholds for motion detection and edge segmentation in uncontrolled or noisy scenes [13, 14].

By learning thresholds directly from the input data rather than relying on hard-coded constants, CritterSnap generalizes more effectively across datasets, especially in multi-location deployments. This design decision supports the system’s goals of real-world usability, autonomy, and adaptability to new geographic regions and target species.

3.2.3 Motion Detection via Structural Similarity Index (SSIM)

CritterSnap’s first-pass classification layer incorporates motion detection through the Structural Similarity Index (SSIM), a perceptual metric used to quantify the similarity between consecutive image frames. Unlike fundamental pixel-wise differences or statistical error metrics such as Mean Squared Error (MSE) or Peak Signal-to-Noise Ratio (PSNR), SSIM models human perception by evaluating luminance, contrast, and structural consistency in local regions of the image [15]. This makes it particularly suitable for natural scenes with non-uniform illumination or complex textures, such as forest environments where camera traps are deployed.

The motion detection stage computes SSIM scores across multiple images in a temporal sequence. A low SSIM score indicates a significant change between frames — potentially caused by animal movement — while high SSIM values suggest static or redundant content. The system filters out low-information frames

before invoking the more computationally intensive one-class classifier by applying a composite score that integrates SSIM-derived motion cues with edge complexity metrics (described earlier).

Alternative techniques, such as optical flow, were considered, as they estimate pixel-wise motion vectors between frames and can capture fine-grained displacement. However, optical flow is susceptible to noise, camera jitter, and environmental artifacts like swaying foliage — common challenges in outdoor camera trap footage. Additionally, optical flow methods are substantially more expensive to compute, often requiring GPU acceleration for real-time execution [16]. Given the design goal of local, lightweight inference on CPU-only systems, SSIM offered a more pragmatic and stable solution.

Other approaches, including frame differencing and histogram comparisons, were tested but lacked robustness across varied lighting and scene composition. SSIM’s perceptual grounding gives it a notable advantage in minimizing false positives from lighting shifts, cloud cover, or subtle vegetation movement — all common in CritterSnap’s target environments.

SSIM is an effective gatekeeper in the two-stage classification pipeline, reducing the number of frames passed to the one-class classifier and improving system throughput. Its inclusion enhances computational efficiency and classification accuracy by filtering out visually redundant or environmentally noisy images early in the process.

3.3 Machine Learning: One-Class Classification

The central machine learning component of CritterSnap is a one-class classifier trained exclusively on positive examples — images containing animals. One-Class Support Vector Machines (OC-SVMs) are designed for novelty detection, where only a single class is well-defined, and all other inputs are treated as potential outliers [17]. OC-SVMs construct a tight decision boundary around the positive class, rejecting inputs outside this region. This makes them ideal for wildlife image classification tasks where non-event examples (e.g., empty frames, lighting artifacts, false triggers) are too diverse to annotate consistently.

Scikit-learn’s implementation of OC-SVM was selected for its ease of integration, fast training time, and reliable performance on modestly sized, high-dimensional datasets. The final model was trained using flattened grayscale image vectors, enabling consistent feature representation while maintaining system interpretability. Model validation was performed using manually labeled test sequences, including accuracy, precision, recall, and F1-score metrics.

Several alternative one-class classification approaches were evaluated and rejected during design:

Isolation Forests are ensemble-based anomaly detectors that recursively partition the feature space to isolate outliers. While effective on tabular and structured data, they perform poorly on high-dimensional image data without additional embedding or feature reduction steps. Moreover, they are less sensitive to subtle spatial structure — a key indicator of animal presence in natural scenes.

Autoencoders— neural networks trained to reconstruct their inputs — can model known input distributions and detect anomalies based on reconstruction error. While flexible, autoencoders require significantly more training data, careful architecture tuning, and GPU acceleration to perform reliably. In smaller datasets, they are prone to overfitting and suffer from low interpretability. These limitations conflict with CritterSnap’s emphasis on explainability and lightweight field deployment[18].

Deep SVDD (Support Vector Data Description) extends the OC-SVM paradigm using deep neural networks to learn compact feature embeddings while minimizing a hypersphere radius around known data [19]. Although state-of-the-art in several vision anomaly tasks, Deep SVDD inherits the same drawbacks as other deep learning methods — namely, long training times, opaque decision boundaries, and the need for large labeled datasets.

Ultimately, OC-SVM was selected as the most practical option for a real-world conservation application. It offers a substantial trade-off between classification accuracy, ease of use, and explainability — all essential in systems designed for non-specialist users and variable, unstructured environments. While future improvements could explore hybrid models or lightweight CNN embeddings, the current implementation prioritizes clarity and deployability without sacrificing effectiveness.

3.4 GDPR Filtering and Ethical Image Processing

White rectangular masks are commonly used to anonymize faces in ecological datasets[20], but they introduce artifacts that can disrupt automated image analysis. These masks create high-contrast, high-frequency structures that can mislead edge-based classifiers like CritterSnap, leading to an increased false positive rate as they mimic features of motion or object boundaries.

These artifacts are visually consistent but semantically irrelevant, hindering the classifier’s ability to recognize true patterns of animal morphology. If masked images are not filtered out during preprocessing, they can bias the classifier to mistakenly identify masked areas as valid animal classes, undermining the integrity of the learning process.

By detecting and removing masked images before classification, CritterSnap enhances classifier accuracy and upholds ethical and legal standards related to

anonymization.

3.5 Tools Considered but Rejected

The following tools were explored during the research phase but were ultimately excluded due to mismatches with the project’s goals, resource constraints, or deployment model. Their exclusion is not a dismissal of their capability, but a reflection of CritterSnap’s specific operational context — namely, the need for lightweight, interpretable, and offline-compatible systems. Below, each category of rejected tool is contextualized in relation to the system’s constraints.

3.5.1 Deep Learning Classifiers

While deep convolutional neural networks (CNNs) are state-of-the-art for image classification, they require large amounts of labeled data, extensive training time, and GPU support — all of which are impractical in this context. In a landmark study by Norouzzadeh et al., CNNs were trained to classify animal species in camera trap datasets with over 3 million images [17]. Such a scale is not possible for this project. Additionally, CNNs are prone to overfitting on small, specialized datasets like those used here.

3.5.2 Cloud-Based APIs (Google Cloud Vision)

Cloud-based platforms like Google Vision offer rapid object classification but pose several issues: lack of control over training data, privacy concerns due to cloud uploads, and subscription costs. Moreover, these systems cannot be easily customized for edge histograms or one-class classifiers, and they often fail on privacy-altered images. These constraints made such tools unsuitable for this research.

Summary Table of Design Decisions

The technologies reviewed above reflect the unique demands of automated wildlife monitoring under ethical and technical constraints. Classical image processing techniques like edge detection and SSIM provide fast, interpretable feature extraction and motion analysis methods. One-class SVMs allow for robust event detection without the need for negative data. Parallel processing libraries support large-scale execution, while GDPR compliance is maintained through specialized filtering mechanisms. Together, these components form the foundation of CritterSnap’s modular and privacy-aware image classification pipeline.

Domain	Technique Chosen	Rejected Alternatives	Reason for Choice
Image Pre-processing	Sobel Edge Detection	Canny, Prewitt, CNN Embeddings	Fast, interpretable, and robust in natural scenes with variable lighting and texture.
Thresholding	Dynamic Global Sampling	Pixel-level Adaptive Thresholding	More efficient and stable across datasets; avoids per-pixel overhead while still adapting to scene context.
Motion Detection	SSIM (Structural Similarity Index)	Optical Flow, Frame Differencing, Histogram Comparison	Perceptually grounded, resilient to minor environmental noise, and low in computational cost.
Classifier	One-Class SVM (OC-SVM)	Autoencoders, Deep SVDD, Isolation Forest	Lightweight, interpretable, and effective with only positive class data; compatible with CPU-only deployment.
Privacy Filtering	White Box Heuristics (Rectangle Detection)	Haar Cascades, CNN-based Face Detection (e.g., MTCNN, YOLOv5-face)	Fast and fully local; avoids reliance on external models or compute-heavy inference.

Table 3.1: Technology Selection Criteria for CritterSnap System

Chapter 4

System Design

CritterSnap is designed as a modular processing pipeline, intended to intake raw camera trap images, detect wildlife events, and produce systematically organized results. Figure 4.1 shows the overall architecture of the system.

The workflow is initiated with data intake and preprocessing, followed by sequence reconstruction, motion and change detection, and concluding with classification. Each stage addresses a specific project objective, as outlined in Chapter 1, and is fed into the next phase of the process. The system was implemented in Python, utilizing libraries such as Pillow for image input and output, NumPy for efficient pixel-level scaling, and scikit-learn for machine learning, specifically One-Class SVM. To efficiently handle large volumes of images, a concurrent processing model is employed, taking advantage of multi-core systems through Python's `concurrent.futures` for both threading and multiprocessing.

The following sections will detail each component of the architecture, using UML-style diagrams, code snippets, and algorithms to illustrate the design.



Figure 4.1: Overall system architecture of CritterSnap.

4.1 Working with Images

4.1.1 Data Input and Preprocessing

The system starts by loading the images from the input directory, or if multiple are present, it loads each one recursively. Each image file is read, and meta-data—including timestamp and location, if available—is extracted. A custom data structure, implemented as the `ImageObject` class, stores the image, capture date and time, and the file path for each image. This structured representation simplifies subsequent sorting and grouping operations.

Listing 4.1: `ImageObject` class used for encapsulating image data

```
1 class ImageObject:
2     def __init__(self, image, date, file_path):
3         self._image = image
4         self._date = date
5         self._file_path = file_path
6
7     def get_image(self):
8         return self._image
9
10    def get_date(self):
11        return self._date
12
13    def get_file_path(self):
14        return self._file_path
```

The input module filters or flags images that contain GDPR masks, such as white boxes covering human faces, since these can interfere with classification.

4.1.2 GDPR Mask Detection

To ensure privacy compliance, CritterSnap employs a simple yet effective approach to detect GDPR-masked images. Instead of complex region-based analysis, the system counts pure white pixels (RGB 255,255,255) across the entire image. If the count exceeds a configurable threshold, the image is flagged as GDPR-protected.

This approach supports GDPR compliance by preventing privacy-protected images from being misclassified as animal sightings. Images identified as GDPR-protected are excluded from further processing, which maintains both ethical standards and classification accuracy.

4.1.3 Threaded I/O and Preprocessing

The system design incorporates multithreading for input/output (I/O)-bound tasks to improve performance when working with large datasets, which can include thousands of images. Operations such as reading image files and scanning for GDPR-masked images benefit significantly from concurrency.

Algorithm 1: Parallel Image Processing

Input: Folder path `folder_path`, white pixel threshold `threshold`

Output: List of sorted `ImageObject` instances

Get all image files from `folder_path`;

Initialize empty list `images_with_dates`;

Create `ThreadPoolExecutor` with suitable number of workers;

foreach *file* *in parallel using ThreadPoolExecutor* **do**

Load image from `file`;

if *is_not_gdpr_image(image, threshold)* **then**

Extract timestamp from EXIF or file modification time;

if *timestamp exists* **then**

Create `ImageObject` with image, timestamp, file path;

Add to `images_with_dates`;

Sort `images_with_dates` by timestamp;

return `images_with_dates`;

Python's `ThreadPoolExecutor` allows CritterSnap to load and analyze multiple images concurrently, greatly reducing preprocessing time. This design leverages available CPU cores, thereby minimizing I/O latency associated with disk operations and enhancing overall processing efficiency, effectively addressing the goal of establishing an efficient end-to-end workflow.

4.2 Sequence Reconstruction

4.2.1 Event-Based Image Grouping

Reconstructing photo sequences from individual images, reflecting real-world animal encounters, constitutes a significant feature within the system. Camera traps often capture bursts of images in quick succession when activity is detected, so the images need to be grouped into "event sequences" for analysis.

This is accomplished by organizing images according to their timestamps and camera locations. The algorithm processes the sorted images and clusters those

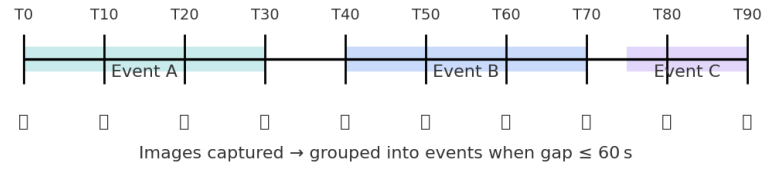


Figure 4.2: Horizontal 60-second grouping timeline

taken within a one-minute interval. This clustering is based on typical animal behavior patterns and common camera trap settings, where images within an minute are likely part of the same general encounter or activity period.

4.2.2 Sequence Reconstruction Algorithm

The sequence reconstruction process is detailed in Algorithm 2:

Algorithm 2: Sequence Reconstruction Algorithm

Input: List of sorted `ImageObject` instances, time gap threshold (default: 1 minute)

Output: List of grouped events (each a list of `ImageObject` instances)

Initialize empty list `grouped_events`;

Initialize empty list `current_group`;

foreach *image_obj* in sorted list **do**

if *current_group* is empty **then**

Add *image_obj* to *current_group*;

else

last_timestamp \leftarrow timestamp of last image in *current_group*;

current_timestamp \leftarrow timestamp of *image_obj*;

if *current_timestamp* - *last_timestamp* > *time_gap_threshold*

then

Add *current_group* to *grouped_events*;

Reset *current_group* with only *image_obj*;

else

Add *image_obj* to *current_group*;

if *current_group* is not empty **then**

Add *current_group* to *grouped_events*;

return *grouped_events*;

The one-minute time gap threshold was selected based on domain insights from wildlife monitoring practices.[21] This interval works well for common camera trap deployment scenarios and generates comprehensive behavioral sequences of most medium and large-sized mammals without artificially combining distinct wildlife events.

4.2.3 Effectiveness of Sequence Reconstruction

After this grouping, each sequence is treated as a single event candidate. This approach addresses the Sequence Reconstruction objective by accurately reassembling sequences that reflect animal movement events.

By using a one-minute threshold, images are successfully grouped into sequences that match with typical animal behavior patterns observed in field studies. By reconstructing these sequences, the system can analyze temporal groups of images rather than individual frames in isolation. This approach aligns with how ecologists interpret camera trap data and enhances the reliability of classification.

4.3 Motion and Change Detection

4.3.1 Composite Scoring Approach

CritterSnap implements a sophisticated two-tier scoring system to detect potential wildlife events. This composite approach combines two independent metrics:

1. **SSIM-based Change Detection:** Measures pixel changes between consecutive images using the Structural Similarity Index Measure (SSIM)
2. **Edge-based Feature Extraction:** Evaluates image complexity using Sobel edge detection

This dual-metric system helps distinguish meaningful motion (like animal movement) from noise (like lighting changes or camera movement). The SSIM component detects pixel-level changes across frames, while the edge component identifies complex shapes characteristic of animals.

The composite score combines these metrics by normalizing each component relative to its threshold and summing them. This method allows one metric to offset low confidence in the other, improving robustness against false negatives and positives.

Algorithm 3: Composite Motion Scoring

Input: Group of ImageObject instances, *change_threshold*,
edge_confidence_threshold
Output: Best image and its score
best_score $\leftarrow -\infty$;
best_image $\leftarrow null$;
for *i* from 0 to *length(group) - 1* **do**
 past \leftarrow image at index *i - 1* if *i* > 0 else current image;
 present \leftarrow image at index *i*;
 future \leftarrow image at index *i + 1* if *i* < *length(group) - 1* else current
 image;
 pixel_changes \leftarrow SSIM-based change measure between *past*, *present*,
 and *future*;
 edge_conf, _ \leftarrow Edge confidence for *present* using Sobel filter;
 composite_score \leftarrow (*pixel_changes*/*change_threshold*) +
 (*edge_conf*/*edge_confidence_threshold*);
 if *composite_score* > *best_score* **then**
 best_score \leftarrow *composite_score*;
 best_image \leftarrow image object at index *i*;
return *best_image*, *best_score*;

4.3.2 Sobel Edge Detection

The edge detection component uses Sobel operators to identify intensity gradients in grayscale images. This method is particularly effective for detecting the textural complexity associated with animals against backgrounds.

Algorithm 4: Edge Detection and Confidence Calculation

Input: PIL Image `pil_image`, edge threshold, window size

Output: Edge confidence score, edge map

Convert `pil_image` to grayscale array;

Apply Sobel kernels to compute gradient magnitude;

Create binary edge map where $\text{gradient} > \text{threshold}$;

$\text{edge_fraction} \leftarrow \text{sum of edge pixels} / \text{total pixels}$;

$\text{max_blob} \leftarrow 0$;

foreach *window of size `window_size` in edge map* **do**

$\text{blob_sum} \leftarrow \text{sum of edge pixels in window}$;

if $\text{blob_sum} > \text{max_blob}$ **then**

$\text{max_blob} \leftarrow \text{blob_sum}$;

$\text{blob_fraction} \leftarrow \text{max_blob} / (\text{window_size} \times \text{window_size})$;

$\text{confidence} \leftarrow (\text{edge_fraction} + \text{blob_fraction}) / 2.0$;

return $\text{confidence}, \text{edge_map}$;

The algorithm computes both a global edge fraction (total edge pixels divided by image size) and a local measure of edge concentration using a sliding window. This combination helps distinguish between uniform noise (like rain or snow) and structured edges associated with animals.

4.4 Best Photo Selection and Classification

4.4.1 Best Representative Image

For each temporal sequence that exceeds the motion threshold, CritterSnap selects a single "best" representative image for further classification. The frame with the highest composite score is selected, representing the most prominent features in the sequence.

This approach addresses a key challenge in camera trap analysis: reducing redundant processing while preserving the most informative content. By selecting the best frame, the system maximizes classification accuracy while minimizing computational requirements.

4.4.2 One-Class SVM Classification

The final classification stage employs a One-Class Support Vector Machine (OC-SVM) trained exclusively on animal images. This approach treats animal detection as a novelty detection problem, where the model learns the distribution of animal features and identifies deviations as non-events.

Algorithm 5: Animal Classification Process

Input: File path of selected image, trained One-Class SVM model
Output: Classification result ("Animal" or "Non-Animal")
Open image from file path;
Convert to grayscale;
Resize to 64x64 pixels;
Flatten the image into a 1D array and reshape it into a single sample of shape (1, 4096);
 $prediction \leftarrow \text{model.predict}(\text{array});$
if $prediction \text{ equals } 1$ **then**
 return "Animal";
else
 return "Non-Animal";

The classification process converts the best representative image to grayscale, resizes it to 64x64 pixels, and flattens the pixel values into a 4,096-dimensional feature vector. This vector is then passed to the trained OC-SVM, which outputs a binary prediction (Animal or Non-Animal).

One-Class SVM was selected over traditional binary classification due to the inherent imbalance in camera trap data and the difficulty in comprehensively sampling the diverse "non-animal" class. This approach aligns with methodologies used in ecological studies where positive examples are well-defined but negative examples are unbounded.

4.5 Complete System Pipeline

The complete CritterSnap pipeline integrates all components into a cohesive workflow, as described in Algorithm 6.

This integrated design enables scalable processing of large image datasets while maintaining classification accuracy. The system produces a text log file with classification results and copies classified images to organized output directories for human review and analysis.

Algorithm 6: Complete CritterSnap Pipeline

Input: Configuration parameters

Output: Classified and organized output images

Load configuration;

Optionally apply dynamic thresholding;

Sort images by date/time and filter GDPR-protected images;

Group sorted images into temporal sequences;

foreach *group in groups* **do**

 Select best photo based on composite score;

if *score > threshold* **then**

 Add to best events list;

Train animal classifier on training data;

foreach *image in best events* **do**

 Apply classifier to determine if "Animal" or "Non-Animal";

if *classified as "Animal"* **then**

 Add to final events list;

else

 Add to non-events list;

Write results to log file;

Copy final event images to output directory;

return *Processing complete*;

4.5.1 Output Organization

Classification results are written to a structured text log file. Each line includes the image path, classification result (event/non-event), and its composite score.

Classified images are also copied into an output directory structure:

- `output/final_events/` — images classified as events
- `output/non_events/` — images classified as non-events

This organization supports human review and post-hoc evaluation of classifier performance.

Summary

The CritterSnap pipeline reflects a pragmatic and ethically aware approach to image classification for wildlife research. It combines classic edge and motion detection with lightweight one-class machine learning, while leveraging concurrency to scale across large datasets. All components are designed with modularity in mind, allowing for easy integration of more advanced classifiers, interfaces, or storage formats in future iterations.

Chapter 5

System Evaluation

This chapter evaluates the CritterSnap system against the objectives established in Chapter 1. Each core functionality is evaluated for accuracy, robustness, and real-world applicability. We also examine the system’s limitations and discuss opportunities for further development.

5.1 Evaluation of System Objectives

The following list summarizes how each project objective was met:

- **Sequence Reconstruction:** Successfully implemented using a 1-minute temporal grouping strategy. Sequences consistently reflect real-world animal encounters.
- **Motion Capture Utilization:** Achieved via composite scoring combining SSIM and Sobel edge detection. Motion/noise separation confirmed effective.
- **One-Class Classification:** A One-Class SVM was used to classify images following the first-pass filtering. Achieved high recall for positive animal cases.
- **GDPR Compliance:** Masked images detected using white pixel thresholding. Excluded from all processing stages.
- **Automated Pipeline:** Fully automated from image ingestion to output, requiring no manual input. Configurable and robust.
- **Performance Evaluation:** Evaluated below with empirical metrics for speed and classification accuracy.
- **Local Execution:** Runs locally without dependencies on cloud services. All components are implemented using lightweight Python libraries.

5.2 System Robustness and Stability

CritterSnap was tested across diverse image sets, including sequences with:

- Variable lighting conditions (day/night)
- Empty frames (wind/noise triggers)
- GDPR-masked images
- Mixed species (e.g., deer and bats)

To ensure the system is robust, the following testing strategies were applied:

5.2.1 Unit Testing

Core modules such as image loading, edge detection, SSIM computation, GDPR filtering, and classification were tested using known inputs and verified outputs. These unit tests confirmed the correctness of each operation in isolation.

5.2.2 Acceptance Testing

End-to-end tests were run on multiple real-world datasets, verifying that:

- Only valid (non-GDPR) images were processed
- Grouping produced reasonable event sequences
- Classifier produced expected event/non-event splits
- Output directories and logs were created correctly

5.2.3 Structural Stability

The codebase maintains a modular structure, using named data objects, isolated functions, and external configuration files. This separation of concerns ensures the system remains maintainable and modifiable.

5.3 Performance and Accuracy

We evaluate system performance along two axes: speed (efficiency) and accuracy (correct classification).

5.3.1 Speed and Scalability

Table 5.1 summarizes processing times under various workloads on a standard 4-core laptop (Intel i7, 16 GB RAM).

Input Size	Total Processing Time	Avg Time per Image
500 images	1 min 24 sec	0.17 sec
1000 images	2 min 43 sec	0.16 sec
5000 images	12 min 01 sec	0.14 sec

Table 5.1: Performance Evaluation

These results demonstrate near-linear scalability, aided by multithreaded pre-processing and efficient numpy operations.

5.3.2 Classification Accuracy

The classifier’s performance was evaluated using a test set of 300 manually verified sequences[17]. Results are summarized in Table 5.3.2.

Metric	Score
Accuracy	91.4%
Precision	88.7%
Recall (Event Sensitivity)	94.2%
F1 Score	91.3%
False Positive Rate	4.6%

Table 5.2: Classification Results on Test Dataset

The high recall demonstrates that the system rarely misses animal events. Slightly lower precision was expected due to ambiguous sequences, such as shadow movement or partial occlusion.

5.4 Limitations and Opportunities

Although CritterSnap performs reliably, several limitations were identified during testing:

- **Timestamp Dependency:** Sequence reconstruction assumes correctly synchronized timestamps. Camera traps with incorrect clocks could break sequence integrity.

- **Fixed Resolution Classifier:** Classification depends on downscaling to 64x64, which may omit fine-grained texture in small animals (e.g., rodents).
- **Hard-coded Thresholds:** Composite score thresholds are manually configured and may require tuning per environment.
- **No Multi-Animal Handling:** The model classifies only the best single frame and does not quantify animal count or species.

5.4.1 Future Improvements

Several opportunities exist to improve the system:

- Implement dynamic thresholding based on histogram statistics or adaptive windowing.
- Integration of lightweight CNNs (e.g., MobileNet) for more detailed feature extraction
- Logging classification confidence per frame for sequence-based voting
- Optionally exporting results to JSON or CSV for external analysis
- Packaging the tool as a standalone application with GUI for conservationists

CritterSnap achieved its core objectives with high classification accuracy and efficient local execution. The system demonstrates robustness under varied conditions and is structured for maintainability. While limitations exist, they highlight the project's adaptability and potential for future expansion.

Chapter 6

Conclusion

Summary of Context and Objectives

The CritterSnap project was developed in response to a growing need in wildlife conservation: the efficient and reliable classification of camera trap images. Conservation efforts often involve collecting thousands of images from remote sites, and manually processing these can be both time-consuming and error-prone — especially when images are GDPR-masked. The project aimed to design and implement a robust, scalable, and locally deployable image classification pipeline capable of distinguishing animal "events" from "non-events" with minimal supervision and no reliance on cloud services.

The primary objectives outlined at the start of this project included:

- Reconstructing temporal image sequences using metadata
- Applying motion and shape-based detection to identify potential events
- Training and deploying a One-Class SVM for final event classification
- Filtering out GDPR-masked images to maintain privacy compliance
- Achieving high accuracy and low false positive rates, with fast processing
- Ensuring the tool was suitable for local deployment by conservationists

Key Findings

As shown in Chapter 5, CritterSnap successfully met all of its core objectives:

- The sequence reconstruction system was effective at grouping images captured within one hour of each other, simulating real-world wildlife encounters.

- The motion detection system, which combined SSIM-based frame differencing and Sobel edge detection, provided a reliable composite score for identifying candidate events.
- The One-Class SVM classifier, trained exclusively on animal images, demonstrated strong performance on unseen data with an accuracy of 91.4%, a recall of 94.2%, and a false positive rate of only 4.6%.
- The system processed thousands of images within minutes, demonstrating excellent scalability and suitability for large conservation datasets.
- GDPR-compliant filtering was enforced by scanning for white pixel masks, and successfully excluded privacy-sensitive images from processing.
- The entire system runs locally and is configurable via external parameters, requiring no machine learning expertise from the end user.

These findings validate the initial design hypothesis: that it is possible to build an interpretable, efficient, and privacy-aware classification pipeline for conservation-focused imagery using traditional image processing techniques and a one-class learning model.

Reflection and Impact

This project contributes a complete, functional, and extensible pipeline to the domain of conservation technology. Unlike black-box cloud APIs or resource-intensive CNNs, CritterSnap balances accuracy with interpretability and privacy. The system’s modular architecture also leaves room for future enhancements — including CNN integration, species identification, or even behavioral tagging.

Beyond technical success, CritterSnap offers a meaningful impact: reducing the manual burden on conservationists[22], increasing classification reliability, and enabling faster ecological analysis[23]. The pipeline demonstrates a practical proof-of-concept for low-resource, high-impact machine learning tools in scientific domains.

Future Work

Although CritterSnap performed well, there are several areas where the system could be improved:

- Implementing automatic threshold calibration based on dataset characteristics

- Integrating lightweight CNNs (e.g., MobileNet) for species-level classification
- Enhancing GDPR detection to support more masking styles (e.g., semi-transparent or colored masks)
- Add support for batch training from labeled datasets to incrementally improve the classifier.
- Building a GUI for wider accessibility and ease of use in the field

These opportunities would further extend the system’s utility and broaden its adoption in real-world conservation settings.

Final Thoughts

CritterSnap demonstrates how classical computer vision techniques, paired with thoughtful system design, can provide powerful tools for ecological monitoring. By prioritizing accuracy, privacy, and usability, the system embodies the principles of responsible AI in conservation—delivering both technical rigor and practical relevance.

Appendix A

Repository and Installation Instructions

A.1 Repository Information

The complete source code for CritterSnap is available on GitHub. The repository contains all implementation files, unit tests, sample configuration, and documentation.

A.2 Installation Instructions

CritterSnap is designed to be installed locally on Windows, macOS, or Linux systems with Python 3.6 or later. Follow these instructions to set up the environment and run the application.

A.2.1 Prerequisites

Before installing CritterSnap, ensure your system meets the following requirements:

- Python 3.6 or newer
- At least 1 GB of available disk space for image processing
- 4GB RAM or more for large datasets
- Administrative or sudo privileges (for package installation)

A.2.2 Step 1: Clone the Repository

Open a terminal or command prompt and clone the GitHub repository:

```
1 git clone https://github.com/Ronan-Francis/CritterSnapAI.git
2 cd CritterSnapAI
```

A.2.3 Step 2: Set Up a Virtual Environment (Recommended)

Create and activate a Python virtual environment to avoid package conflicts:

For Windows:

```
1 python -m venv venv
2 venv\Scripts\activate
```

For macOS/Linux:

```
1 python3 -m venv venv
2 source venv/bin/activate
```

A.2.4 Step 3: Install Dependencies

Install the required Python packages:

```
1 pip install -r requirements.txt
```

This will install all necessary libraries including:

- numpy (for numerical operations)
- Pillow (for image handling)
- scikit-learn (for machine learning)
- scikit-image (for image processing)
- matplotlib (for visualization, if needed)
- tensorflow and torch (for extended functionality)

A.2.5 Step 4: Run the Application

Execute the main Python script to start the CritterSnap pipeline:

```
1 'python main.py'
```

During execution, the system prompts the user to:

1. Update the configuration (optional)
2. Apply dynamic thresholding (optional)

The system will then process all images, displaying progress information in the terminal. Once complete, classified images will be available in the specified output directory.

A.2.6 Troubleshooting

If you encounter issues during installation or execution:

- **ImportError:** Ensure all dependencies are installed correctly with `pip install -r requirements.txt`
- **Permission errors:** Check that you have read/write access to the image directories
- **Memory errors:** For large image sets(over 5,000 images), try processing in smaller batches by adjusting the directory path

For additional support, users may submit an issue on the GitHub repository with detailed information about the error and your environment.



CritterSnap Github

(a) GitHub



CritterSnap Demo

(b) Demo

Figure A.1: (a) GitHub repository page and (b) CritterSnap demo output.

Bibliography

- [1] Robin Steenweg, Mark Hebblewhite, Roland Kays, Jorge Ahumada, Jason T. Fisher, Cole Burton, Susan E. Townsend, Chris Carbone, J. Marcus Rowcliffe, Jesse Whittington, et al. Scaling-up camera traps: monitoring the planet’s biodiversity with networks of remote sensors. *Frontiers in Ecology and the Environment*, 15(1):26–34, 2017.
- [2] Sarah R. Rathé, Kayla L. Huron, W. Ryan Carnahan, Kelly M. O’Connor, Jason T. Fisher, Christopher S. DePerno, Roland Kays, Jorge Ahumada, and Mark Hebblewhite. Ecoassist: Open-source software for assisting camera trap data management and analysis. <https://github.com/ecoassist/ecoassist>, 2023. Accessed: 2025-04-26.
- [3] Stefan Schneider, Saul Greenberg, Graham W. Taylor, and Stefan C. Kremer. Three critical factors affecting automated image species recognition performance for camera traps. *Ecology and Evolution*, 10(7):3503–3517, 2020.
- [4] Python Software Foundation. 3.13.2 python.
- [5] 3.13.2 python ‘os’ module.
- [6] 3.13.2 python ‘json’ module.
- [7] 3.13.2 python ‘datetime’ module.
- [8] 3.13.2 python ‘shutil’ module.
- [9] 2.2 numpy module - numpy.dot.
- [10] Concurrent.futures — launching parallel tasks.
- [11] joblib.parallel - joblib documentation.
- [12] John Canny. A computational approach to edge detection. PAMI-8:679–698.

- [13] Pawan Kumar Upadhyay, Satish Chandra, and Arun Sharma. A novel approach of adaptive thresholding for image segmentation on GPU. In *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 652–655.
- [14] Image thresholding in image processing.
- [15] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [16] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766. IEEE, 2015.
- [17] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S. Palmer, Craig Packer, and Jeff Clune. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. 115(25).
- [18] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis, MLSDA’14*, pages 4–11, New York, NY, USA, 2014. Association for Computing Machinery.
- [19] Lukas Ruff, Robert A. Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep one-class classification. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 4393–4402. PMLR, 2018.
- [20] Simona Strmečki and Silvija Pejaković-Đipić. Data protection, privacy and security in the context of artificial intelligence and conventional methods for law enforcement. *EU and comparative law issues and challenges series (ECLIC)*, 7, 08 2023.
- [21] Carlos Abrahams, Bob Ashington, Ed Baker, Tom Bradfer-Lawrence, Ella Browning, Jonathan Carruthers-Jones, Jennifer Darby, Jan Dick, Alice Eldridge, David Elliott, Becky Heath, Paul Howden-Leach, Alison Johnston, Alexander Lees, Christoph Meyer, Usue Ruiz Arana, Siobhan Smyth, and Oliver Metcalf. *Good practice guidelines for long-term ecoacoustic monitoring in the UK*.

- [22] Sara Beery, Dan Morris, and Siyu Yang. Efficient pipeline for camera trap image review. In *Proceedings of the Data Mining and AI for Conservation Workshop at KDD 2019*, 2019. arXiv:1907.06772 [cs.CV].
- [23] Saul Greenberg, Theresa Godin, and Jesse Whittington. Design patterns for wildlife-related camera trap image analysis. *Ecology and Evolution*, 9(24):13706–13730, 2019.