UNIVERSITÉ DU LUXEMBOURG

Interdisciplinary

Space Master Program

MSc Thesis

Spring – Summer 2023

Machine Learning for Classification of Satellite Geolocation Data:
Addressing the Path Segmentation Challenge in Animal Movement Ecology

AUTHOR:

Elliott Wobler

TAGS:

MOVEMENT ECOLOGY, BIOTELEMETRY, EARTH OBSERVATION, MACHINE LEARNING, RNN, LSTM, TIME-SERIES CLASSIFICATION, ANIMAL MIGRATION, BIODIVERSITY, CONSERVATION MANAGEMENT

TABLE OF CONTENTS

ABSTRACT

Knowledge of global animal movement offers insight into our changing planet, and direct observation of patterns from space is an ideal vantage point. Due to hardware miniaturization of animal trackers and satellites (CubeSats), increasing numbers of geolocation records are becoming available. Ecologists, biologists, and conservationists apply this data in their research and initiatives, but robust methods for automated classification of the data are lacking. In order to quantify behavioral changes at scale for the study and stewardship of nature, a system is needed that can automatically segment and label movement states. Such a system can benefit science by reducing the setup time for research, thereby improving resource allocation of people, time, and funding. This manuscript explores the viability of machine learning models to address the challenge of segmenting active migration from summer or winter range residency. Recurrent neural network (RNN) and long short-term memory (LSTM) architectures are both evaluated and compared. Results show encouraging accuracy with F1-scores exceeding 80%, and work is scoped for future optimizations and feature inclusion.

1 INTRODUCTION AND LITERATURE REVIEW

Wildlife monitoring from space is a key component of Earth Observation. NASA's "Internet of Animals" group (IoA) is one of various subcommunities advancing the field. The IoA effort is a collaboration between the Jet Propulsion Laboratory (JPL), Yale University, NASA Ames Research Center, and the USGS Western Ecological Research Center [1]. The group is assessing the need for investment in space-based animal tracking, considering the mandates of various agencies. It plans to propose a role for NASA to assist in meeting these requirements.

The initiative began when the multi-agency Satellite Needs Working Group (SNWG) identified the potential value offered by "increased availability of animal movement data, the dynamic integration with remotely sensed data, and the novel computational tools to support data-informed decision-making" [2]. Along with planet-scale biosphere science, one key goal is ensuring that animal tracking is considered and funded in upcoming NASA decadal surveys.

To this end, the research reported below addresses the need for a specific computational tool defined by IoA members. The sections cover: 1) the problem my method aims to solve and my research questions with background context; 2) the data used during implementation and validation; 3) details on techniques employed to implement my method; 4) results of my method and their possible implications; and 5) a brief conclusion that directly answers my research questions.

1.1 Tracking Animals from Space

The volume of observations collected by spaced-based systems has greatly increased the total amount of global animal tracking data [3]. Records on birds, mammals, amphibians, and reptiles have all seen significant growth, with avian species collectively representing the largest category (see Appendix 2 for details). Adding to the data available today are observations from novel sensors which enable animals to passively sample environmental information [4].

Spaced-based biotelemetry methods with Earth-orbiting satellites have been practiced for several decades and discussed for more than half a century [5]. As with other forms of Earth Observation, satellites enable a global view into some of the planet's most remote regions. Since the 1980s, mammals such as caribou *(Rangifer tarandus)* and polar bears *(Ursus maritimus)* have been studied in isolated northern environments with the Argos Data Collection and Location System (see Appendix 3 for the Argos system architecture) [6]. Investigations of these and other species are crucial for understanding the effects of climate change and human activity on vulnerable ecosystems.

Today, one of the primary resources for accessing animal movement data is Movebank [7]; a free, online database hosted by the Max Planck Institute of Animal Behavior. Movebank's community of 3,781 unique data owners has contributed to a total of 4.8 billion records from 1,288 distinct taxa, using various tracking systems including Argos and GPS [ibid]. NASA's 3D visualization of the Arctic Animal Movement Archive – tens of millions of location records hosted through Movebank – helps to show the scale of this rapidly growing dataset [8]. Contributors to open-source software projects have also written specialized tools that accelerate Movebank data retrieval and analysis in locally cloned databases [9].

In 2002, a project began called the International Cooperation for Animal Research Using Space (ICARUS). Responsible wildlife tracking protocol forbids using trackers that exceed 5% of an animal's total body mass, so many species are still untracked [10]. Therefore, one of the aims of ICARUS is to develop miniaturized tracker technology [11]. The smaller that tracking devices become, the more global species can be ethically tracked.

Prototype tracker devices were developed for ICARUS with multi-sensor data loggers [12]. In addition to reporting GPS location, measurements were made from accelerometers, magnetometers, and temperature gauges. Wind speed and direction could be monitored too, allowing for more environmental context [11] (see Appendix 4 for additional tracker information).

The ICARUS space segment payload was co-developed by DLR (the German Aerospace Center) and Roscosmos (the Russian Space Agency). It was hosted on the Russian Segment of the International Space Station (ISS) in December 2019, and operations began in later months after testing and validation (see Appendix 3 for the ICARUS system architecture) [12].

Following the Russian invasion of Ukraine in early 2022, ICARUS data transmissions from the ISS were stopped in March [13]. Despite those events, early ICARUS data continued to be utilized [14]. A "Digital Museum of Animal Lives", hosted at AnimalLives.org, showcases some of the data collected and highlights the lasting promise of ICARUS [15].

In June of 2023, an experimental ICARUS CubeSat was launched to validate capabilities for the system launching in 2024 [16]. Since the ISS orbital inclination restricts it to latitudes between +51.6° N and -51.6° S [17], the coverage offered by Sun-synchronous CubeSat orbits (SSO) is a significant architectural improvement. Global coverage enables monitoring Earth's polar regions which are at a higher risk from the effects of climate change [16].

Additional space-based wildlife tracking missions are also under development. The University of Warwick supports the multi-year "WUSAT Programme" which manages CubeSats designed by students to complement ICARUS efforts [18]. A further iteration of NASA's GRACE and GRACE-FO missions has also been proposed. GRACE-i (short for GRACE-ICARUS) aims to support the parallel tasks of monitoring global water resources as well as biodiversity [19].

1.2 The Value of Animal Movement Ecology

From protecting songbird migration flyways [20], marine ecosystems [21], and poaching targets [22]; to monitoring the spread of zoonotic diseases [23] and predicting earthquakes through behavioral cues [24]; global animal tracking data supports many use cases.

Studies have shown that mammal movements in regions with a greater human footprint were one-half to one-third the extent of their movements in areas with less human activity, on average [25]. Since animal mobility is fundamental for species survival and ecosystem health [26], understanding variations in movement – along with habitually taken routes – is a key part of comprehensive conservation management.

Once habitual routes are confirmed, protected movement corridors can be established. These can help protect animal activity from human-induced disruptions that are more frequent in urban landscapes [27]. Cities around the world – from Los Angeles [28] to Kenya [29] – have shown that corridors are beneficial not only for safer animal transit but also as natural habitat extensions. Traditional ecological knowledge from

Indigenous communities can also complement tracking data to ensure the best outcome for ecosystems [30] [31].

Disruptions in usual movement behavior can warn of wider environmental changes. One example is the feedback loop caused by displaced North American beavers *(Castor canadensis),* a well-known ecosystem engineer that also serves as a keystone species. When rising temperatures force the beavers to seek a more comfortable northern climate, ponds they create while constructing their dams then melt the surrounding permafrost [32]. As permafrost melts, it releases methane, and contributes to greenhouse gas emissions. This example serves to illustrate one of many biosphere interconnections relevant to changing biodiversity as well as climate conditions.

1.3 The Problem of Animal Movement Segmentation

Ecological science advancement depends on tools that work at scale. This is due to increasing Earth Observation data volumes and represents a growing opportunity from recent decades (Appendix 2). Specific to the case of animal movement, efficient and flexible methods are needed to classify billions of satellite geolocation records with global distribution.

One type of classification is movement path segmentation. Path segmentation refers to classifying sequential path waypoints and discretizing continuous trajectories into separate segments. In the context of annual avian migration – the specific focus of my research – I selected four high-level segment classes: summer home range, active migration, winter home range, and the migration back.

Once this segmentation can be performed at scale without relying on manual human classification, ecologists can devote larger portions of their time to answering specific scientific questions. Examples of such questions include but are not limited to:

How much of a species' home range lies within a protected zone? Could some proposed construction project negatively impact breeding grounds? What is the vegetation index value of winter versus summer home range? How are animal migration patterns shifting due to climate change?

If the tedious segmentation task is automated across a variety of species, resources like time, personnel, and funding can be allocated more towards addressing these questions.

1.3.1 Current Methods for Addressing Segmentation

Current approaches to animal movement segmentation and classification work in some cases but typically lack in either flexibility, precision, or efficiency at scale. Several methods exist that require human supervision or are relatively sensitive to user input settings, so in practice are not always more useful than manual validation (see Table 1).

Other methods are sometimes very computationally expensive, or otherwise not well-suited to the challenges posed by big data. Table 1 below outlines existing popular methods and describes some of their operational advantages and disadvantages.

Method	Description	Advantages	Disadvantages
Hidden Markov Model (HMM) [33] [34]	Probabilistic approach which estimates hidden behavioral states and transitions throughout a movement path; assumes future state depends on current state, not previous states (Markov property)	Capturing dependencies between two consecutive states; able to handle uncertainty in noisy or missing data; flexible enough to also incorporate environmental features	Limited precision due to Markov property assumption; sensitive to input settings; computationally expensive; complex implementation
Segclust2d [35]	Spatial clustering approach originally designed for segmenting 2D animal movement data into homerange and behavioral states	Simple and effective for straightforward 2D segmentation; computationally efficient; useful for less complex movement; includes a minimum segment length setting	Less effective for complex patterns of behavioral transitions; lacks probabilistic detail for uncertainties; can be sensitive to input settings
Piecewise Regression [36]	Linear regression approach that segments movement data into distinct “pieces” by identifying breakpoints where different linear trends are present	Easy to interpret and visualize; finds exact breakpoints where behavioral shifts occur; good for abrupt changes; limits state pattern assumptions	Assumes linear paths within segments; limited insights on complex movement; can miss more gradual behavioral shifts
Behavioral Change Point Analysis (BCPA) [37]	Likelihood-based statistical approach that identifies primarily abrupt shifts, or “change points”, in irregularly-sampled time-series data	Can detect non-linear movement shifts; suitable for a variety of state transitions; statistical approach helps separate random fluctuations vs significant changes	Limited to abrupt changes; stepwise approach may not capture gradual transitions or subtle variations; can be computationally expensive

Table 1: Strengths and limitations of existing methods for animal movement path segmentation

The convergence of increasing satellite data with recent advances in machine learning (ML) offers a promising avenue for addressing the segmentation challenge. Machine learning methods are especially effective with large datasets. They are already used in various ecology applications [38] including imagery classification for aerial observations and camera traps [39]. This indicates that a portion of researchers are familiar with ML tools and possess the technical capacity to adopt ML methods for path segmentation.

One directly relevant ML approach for segmentation was documented in *Overton, et al* [40]. In this project, the researchers used hourly GPS location data obtained from five species of common North American dabbling ducks *(Anatidae)*. They aimed to classify daily activity patterns with eight label categories specific to their selected species: brooding, dead, local, migration, molt-like, molting, nesting, and regional relocation. To do this, the team developed multiple input features composing three main groups: GPS locations, movement history, and remotely sensed habitat characteristics.

The team used Amazon Web Services' SageMaker Studio with Autopilot to automate steps within the machine learning workflow. They selected variations of three model frameworks: Extreme Gradient Descent *("XGBoost");* Stochastic Gradient Descent *("LinearLearner");* and Multi-Layered Perceptron (MLP). Employing automated pipelines allowed them to reduce the number of non-ecological decisions. They compared different model performance metrics using weighted F1-scores to balance precision and recall (terms defined in Section 3.2.4: *"Evaluating Model Performance"*). They reported a maximum F1-score of 95% and deemed their method appropriate for their specific research context.

1.3.2 Proposed Segmentation Solution and Research Questions

Models specifically intended for use with time-series data – namely, recurrent neural networks (RNNs) [41] – have frequently been employed in applications for animal movement predictions [42][43][44]. Recent studies have suggested their potential for comparative trajectory segment analysis [45], inviting a deeper investigation into their efficacy.

My research described below aimed to evaluate RNNs as a candidate for movement path segmentation tasks. My approach emphasized visibility into model configurations and optimizations by using the `PyTorch` framework. I also included a curated set of ablation studies in my results, along with the top-performing model's details.

My approach aimed to use classes and features that generalize across a variety of species. It limited label categories to the four classes outlined above: summer home range, active migration, winter home range, and the migration back. These classes can be applied to a wide array of migratory animals. Similarly, input features to the model were initially limited to movement and time. Environmental and habitat features were left for future iterations so the RNN approach could be reviewed with minimal variables.

In later sections, I describe my research in detail. The conclusion of this manuscript responds to both of my research questions:

- **Question 1:** What accuracy is achievable when segmenting satellite-collected animal movement data with recurrent neural networks?
- **Question 2:** What accuracy is achievable using this method with minimal time and movement features (i.e. excluding environmental data)?

2 DATASET DESCRIPTION

The data used to train the model implemented during the course of this research was provided by members of the Yale University Center for Biodiversity and Global Change (BGC). Data for five species of migratory cranes were shared containing two types of information. One file stored all of the geolocation events data, collected via satellites for various cranes outfitted with tracking devices. Other files were provided for each individual animal, with human-labeled segmentations describing the relative migratory states.

As part of the first exploration actions taken before receiving the data, an interactive web-based tool was built for data investigation. The tool features three-dimensional geospatial visualization with satellite imagery and time controls for navigating animal movement data. It can be used to manually validate movement continuity or gain environmental context on local topography (e.g. mountains and valleys). Source code for this supplementary work is included in Appendix 1, along with links to a video showcase and live hosted webpage.

2.1 Details on the Data used for Model Training

The primary unlabeled data file – hereafter referred to as the “events data” – contained 1.21 Gigabytes of satellite-collected animal tracking data. As stated above, the data represented five species of cranes (see Table 2).

The events data included ~1.7 million geolocation records, each one a trajectory waypoint with additional metadata. Of the 143 columns, only five were directly used:

1. `individual_id` : the unique identifier for the individual animal
2. `species` : the species of the individual animal
3. `timestamp` : the time of the geolocation record (provided in UTC)
4. `lat` : the latitude of the geolocation record
5. `lon` : the longitude of the geolocation record

Complementary labels for migration segments were provided as 182 separate files. These smaller files used the following four columns to define temporal windows and associated `Status` labels:

1. `Individual` : the unique identifier for the individual animal
2. `Species` : the species of the individual animal
3. `Date` : the transition date linking two separate migratory states
4. `Status` : the string label defining the current/new migratory state

In the 182 label files, the `Date` column indicates the beginning or end of a temporal range defining a specific behavioral state, or `Status`. In other words, the `Date` is the transition from one `Status` to another. This enables the events and label data to be merged together. The twelve possible string values for `Status` were: *["Start Fall", "End Fall", "Start Spring", "End Spring", "Presumed Start Fall", "Presumed End Fall", "Presumed Start Spring", "Presumed End Spring", "Start Stopover", "End Stopover", "No migration", "Not enough data"].*

Records with a value of *"Not enough data"* were immediately purged after loading each file. For both the label and event datasets, a unique list of individual animal identifiers was produced and checked against the other dataset. If there were events for individuals without any labels, or labels for individuals without any events, these records were discarded. Following this initial cleanup, *"No migration"* was not represented.

Table 2 below shows the total count of event records broken down by species after preliminary data filtering. The number of individual animals represented from each species is included to show the general taxa diversity within the data. A geospatial visualization was also created with `kepler.gl` to understand the spatial distribution of each species (**Fig. 1**) [46]. Temporal distribution is shown in a data density timeline. The complete time range is 16 February 2011 to 24 March 2021, with most event records occurring between late 2013 and early 2021.

Animal Species	**Binomial Name (genus + species)**	**# of Event Records (waypoints)**	**# of Individual Animals**
Common Crane	*Grus grus*	780,837	17
Demoiselle Crane	*Anthropoides virgo*	173,646	45
White-naped Crane	*Grus vipio*	156,154	8
Black-necked Crane	*Grus nigricollis*	46,932	5
Paradise Crane (or "Blue Crane")	*Anthropoides paradiseus*	506	2

Table 2: Species-level data details after preliminary data cleanup

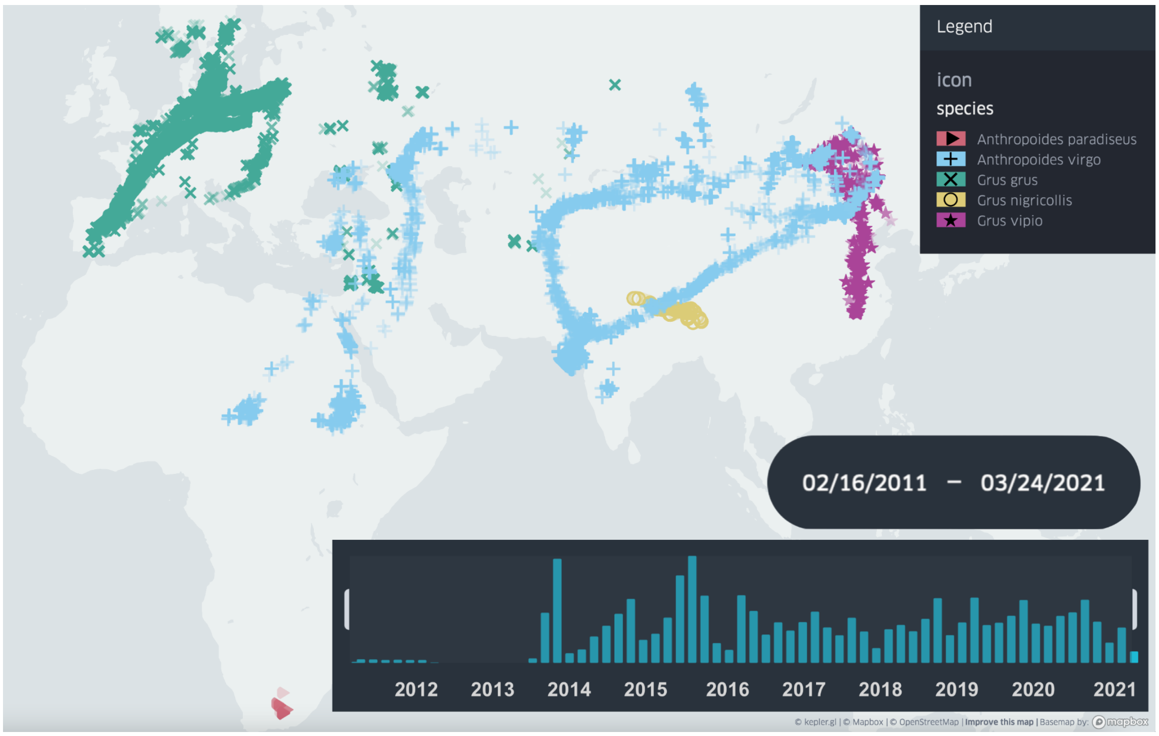


Fig. 1: Visualization of time-series crane locations, colored by species, with data density timeline

Labels with *"Presumed"* values were counted directly as the corresponding class. In theory, a binary `confidence` column could have been used as an input feature, but it was excluded to keep the fields consistent with unlabeled real-world data. An assumption was made that any human-provided labels – regardless of being "presumed" or not – were accurate enough to be included as the ground truth. Stopover labels were also excluded to focus on the primary task, since classifying these secondary states was not part of the research goal.

Following this, the original twelve `Status` values were filtered to: *["Start Fall", "End Fall", "Start Spring", "End Spring"].* When officially merging the events and labels datasets, the implicit labels of *"Winter"* and *"Summer"* were explicitly added based on transition dates. I have included step-by-step implementation details on the merging process in Supplement 1.

Table 3 below provides a mapping of the final class labels to associated behavioral states for the species of cranes represented in the data. Although the class label strings are the same as traditional season names, this is just semantic ecological shorthand. The important information that these strings encode is the generalized crane behavior expectation within each of the four segmentation classes.

Final Class Label	Associated Behavioral State for Cranes
Summer	Breeding
Autumn	Migration (post-breeding)
Winter	Non-breeding
Spring	Migration (pre-breeding)

Table 3: Mapping of segmentation class labels to associated behavioral states for cranes

3 MODEL IMPLEMENTATION

The following sections give an in-depth look at my RNN implementation process, from data preprocessing to final model evaluation. All referenced code was written in Python, aside from bash scripts for the HPC workflow. The machine learning framework employed was `PyTorch`, with common supporting utility modules: `numpy`, `pandas`, and `matplotlib`. Model performance evaluation was done with `scikit_learn`. The complete source code for my research is available at the URLs in Appendix 1**.**

3.1 Data Preprocessing and Feature Engineering

Once all geolocation events were labeled with a class from Table 3, I derived additional time and movement features from the existing columns. I then normalized the feature data and split it into three subsets, a workflow outlined in Fig. 2 below.

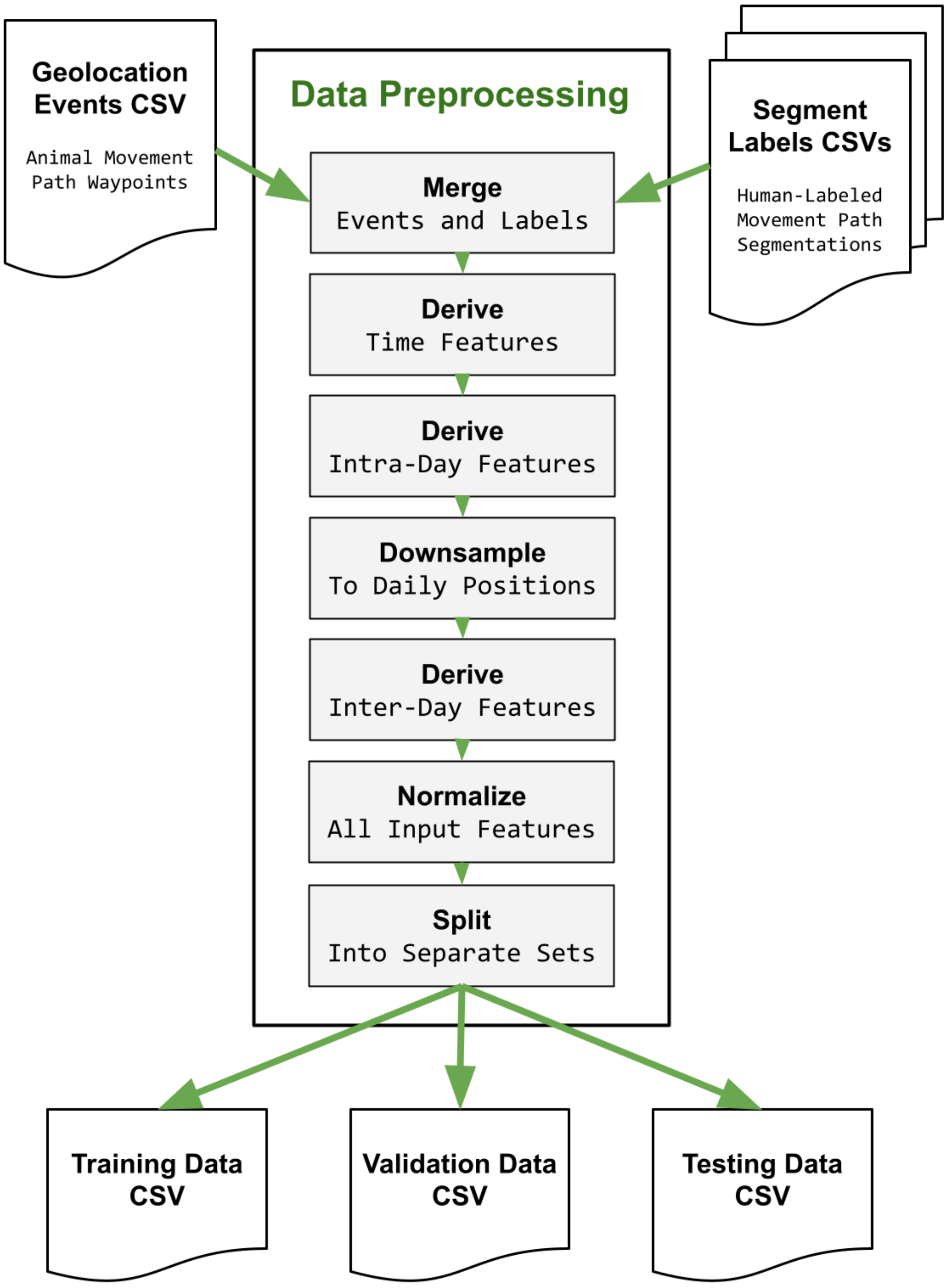

Geolocation Events CSV
Animal Movement Path Waypoints
Data Preprocessing
Segment Labels CSVs
Human-Labeled Movement Path Segmentations
Merge Events and Labels
Derive Time Features
Derive Intra-Day Features
Downsample To Daily Positions
Derive Inter-Day Features
Normalize All Input Features
Split Into Separate Sets
Training Data CSV
Validation Data CSV
Testing Data CSV

Fig. 2: Block diagram overview of the data preprocessing workflow

3.1.1 Deriving Features and Daily Downsampling

The final set of input features can be broadly grouped into: geolocation (latitude/longitude), intra-day movements (daily means), inter-day movements (day-to-day steps), and several time-related features. This subsection details how those features were derived. The full set of column names within the transformed dataset are described in Table 4 below. Indices A-E were the metadata columns and indices 1-14 were the final model's input features.

Time Features

For each record, I parsed the year, month, and day values out of the timestamp. Next, I passed the timestamp to a function for deriving "cyclical time", represented as two new fields: `sin_time` and `cos_time`.

Including these sine and cosine time features allowed me to encode the cyclical nature of seasonality into numerical representations. In contrast to a linear representation of years with integer values (e.g. 2018, 2019, etc.), this ensured that December 31 and January 1 were cyclically adjacent, rather than representing them at opposite sides of a linear spectrum. Learning seasonal relationships between the input data can improve the model's ability to find patterns in animal movement [47].

To represent a calendar year in terms of cyclical time, I calculated the "seconds since the start of the year". I then computed the sine and cosine values of the angle (θ) between the `timestamp` (i.e. seconds since the start of the year) and the "full cycle" (i.e. total seconds in the year). This full cycle was represented as 31,536,000 seconds. In mathematical terms, the calculation was:

$$\theta = 2\pi \bullet \frac{seconds\_since\_year\_start}{31{,}536{,}000}$$

Intra-day Movement Features

After defining the time inputs, I then generated the movement features. Due to the fact that inter-day features required daily downsampling, I handled the intra-day features first in order to capture all of the events.

Following all of the calculations described in this subsection, four new movement features were added to the input records: the distance from the previous location, the velocity between two consecutive locations, the bearing between two consecutive locations, and the turn angle between two consecutive locations. These are all common metrics employed in animal movement analysis [48].

First, I calculated latitude and longitude differences between consecutive rows for each individual's chronological waypoints. Using these `lat_diff` (Δφ) and `lon_diff` (Δλ) values, I then applied the Haversine formula to calculate the spatial differences [49]. In mathematical terms, the calculation was as follows, with R_{Earth} representing the Earth's mean radius in meters (6,371,000 meters) :

$$a = sin^2(\Delta\varphi/2) + cos\,\varphi_1 \bullet cos\,\varphi_2 \bullet sin^2(\Delta\lambda/2) \qquad (1)$$

$$c = 2 \bullet atan2(\sqrt{a}, \sqrt{(1-a)}) \qquad (2)$$

$$d = R_{Earth} \bullet c \qquad (3)$$

Once I obtained the spatial distance between all consecutive waypoints, I also calculated the time differences for consecutive `timestamp` columns. With distances in meters and time deltas in seconds, I computed a `velocity` column using $v = d/t$.

Then, with just the latitude and longitude values, I calculated a `bearing` (θ) column in the range of -180 to 180 degrees:

$$x = cos(\varphi_2) \bullet sin(\Delta\lambda) \qquad (1)$$

$$y = cos(\varphi_1) \bullet sin(\varphi_2) - sin(\varphi_1) \bullet cos(\varphi_2) \bullet cos(\Delta\lambda) \qquad (2)$$

$$\theta = degrees(atan2(x, y))\ \%\ 360 \qquad (3)$$

$$\theta = \theta - 360\ if\ \theta > 180 \qquad (4)$$

The bearing (θ) value was then used to calculate the turn angle (ψ) at each waypoint:

$$r = radians(\theta - roll(\theta, 1)) \qquad (1)$$

$$\psi = atan2(sin(r), cos(r)) \qquad (2)$$

$$\psi = degrees(\psi) \qquad (3)$$

The `year`, `month`, and `day` columns were used to group the data records, and the daily mean was calculated for each of the four new movement features: distance, velocity, bearing, and turn angle. Finally, I merged the calculated intra-day means with the relevant waypoint records.

Inter-day Movement Features

With all of the geolocation records accounted for via the daily means, thereby capturing intra-day activities in new movement summary features, the calculation for inter-day activities began by daily downsampling. Specifically, I filtered the waypoints to only the latest geolocation record of each individual's daily movements.

Alternative methods – such as averaging or calculating the centroid – could also be investigated in future experiments. For my research, the decision was made to maintain the integrity of the underlying geolocation data for each individual. Creating a centroid can misrepresent an individual's actual path by placing them at a coordinate that they never passed through. This concern was the rationale for choosing the daily "last known position", preserving the latitude and longitude values included within the original data.

After reducing the total number of records by daily downsampling, I once again calculated the four movement features between each individual's chronological waypoints. This resulted in eight total movement feature columns: the four mean daily values outlined above, and the four new inter-day values just described. In this way, I represented animal movement activities as numerical input features both *within* and *between* days of each sequence.

Index	Final Data Column Name	Data Type	Data Description
A	individual_id	*string*	unique ID for the individual animal
B	species	*string*	species of the individual animal
C	status	*string*	class label proxy for behavioral state *(Autumn, Winter, Summer, or Spring)*
D	timestamp	*datetime*	time of the geolocation record (UTC)
E	year	*integer*	year parsed from the timestamp
1	*month*	*integer*	month parsed from the timestamp
2	*day*	*integer*	day parsed from the timestamp
3	*sin_time*	*float*	sine of the angle between the timestamp and the cycle (year)
4	*cos_time*	*float*	cosine of the angle between the timestamp and the cycle (year)
5	*lat*	*float*	latitude of the geolocation record
6	*lon*	*float*	longitude of the geolocation record
7	*dist_from_prev_loc*	*float*	inter-day distance traveled between consecutive geolocation records
8	*velocity*	*float*	inter-day velocity between consecutive geolocation records
9	*bearing*	*float*	inter-day bearing between consecutive geolocation records
10	*turn_angle*	*float*	inter-day turn angle between consecutive geolocation records
11	*daily_mean_distance*	*float*	intra-day mean of distance values
12	*daily_mean_velocity*	*float*	intra-day mean of velocity values
13	*daily_mean_bearing*	*float*	intra-day mean of bearing values
14	*daily_mean_turn_angle*	*float*	intra-day mean of turn angle values

Table 4: Mapping of metadata (A-E) and final model input features (1-14) to data types and descriptions

3.1.2 Visual Validation of Input Features per Class

After I derived all of the input features, I manually checked their values to validate that the output ranges looked correct. To assist in this process, I generated a set of histogram visualizations. For each input feature, I created four separate histogram plots; one for each of the four label classes. This resulted in 56 histogram images: $4_{classes} \times 14_{features} = 56_{histograms}$.

Along with the standard histogram visualization, I included the kernel density estimate (KDE) line which helps to make the general trends more interpretable [50]. Due to the data spread, I chose a logarithmic Y-axis scale. This enabled seeing more detail in the less frequent ranges.

Four samples of these histogram visualizations are included below for reference. The turn angle values for both *"Summer"* and *"Winter"* labels are within the expected range of -180 to +180. The daily mean velocities for two different classes are also within the possible range of speeds for the avian species considered.

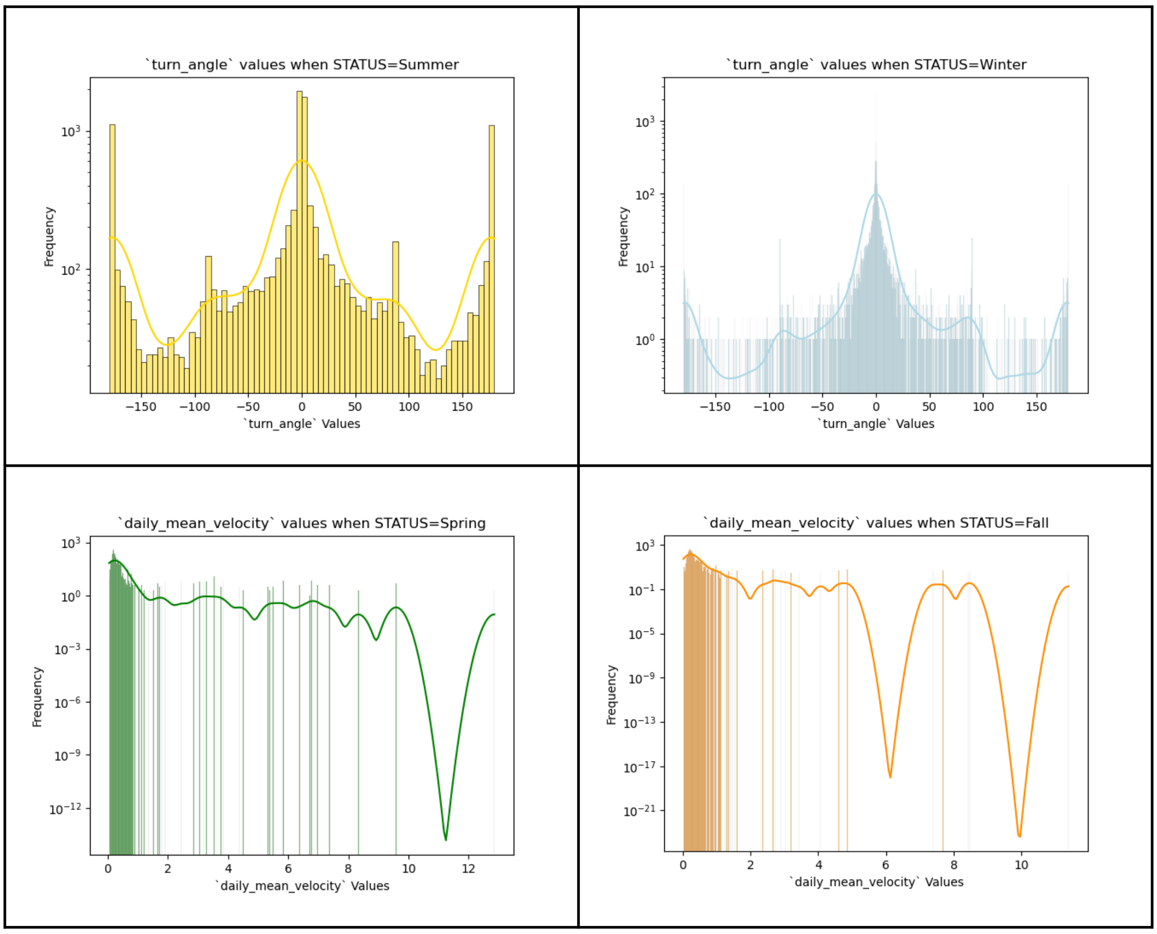


Fig. 3: Example histogram visualizations with KDE lines showing input feature values per label class

3.1.3 Normalizing the Input Features

In ML methods using gradient descent (defined in Section 3.2.2), normalization of input features is a common practice. This is done to ensure that features with varying numerical ranges are scaled consistently, preventing arbitrary bias of the model towards a specific feature. Different normalization techniques are employed for different use cases, including: Z-score scaling, robust scaling, and min-max scaling.

For my research I used min-max scaling across all input features. The minimum and maximum theoretical values were used where possible. The minimum and maximum `month` values were set to 1 and 12, respectively. The range of the `day` feature values was set from 1 to 31. The latitude range was set from -90 to +90 and the longitude range was set from -180 to +180. For other features – such as the distance traveled and mean daily velocity – value ranges were calculated directly from the data.

Once the minimum and maximum values were determined for all of the input features, each of those fields was normalized – resulting in new, scaled data values between 0 and 1.

Manual validation was performed on each of the computed value ranges, ensuring the values were biologically feasible for the five species in question. Specifically, distances and velocities were checked to ensure that values didn't exceed the theoretical range. Future iterations of this research combining different species should take their relative movement speeds into consideration. Trying alternate normalization methods – such as Z-score scaling – might also yield improved results. This is because Z-score scaling is better at handling outliers than the linear transforms inherent to the min-max method [51].

3.1.4 Splitting the Data into Train, Test, and Validation Sets

Splitting input data into separate training, validation, and final testing subsets is also a common practice when developing ML models. The reasons for this are described in detail below in Section 3.2.3. As shown in Fig. 2, this split is done after input features have been derived and normalized.

In straightforward applications where every data row is a distinct entity, these subsets are typically split by defining a target percentage breakdown. For example: 80% of the event rows could be used for model training; 10% could be used for validation during that process; and finally, the remaining 10% of the data could be reserved for testing the performance of the fully trained model.

In my research, a simple percentage split would not suffice. This is because each data row is not a distinct entity, but rather a single waypoint in a sequential movement path that a specific individual performed chronologically. Therefore, in addition to target percentage breakdown values, other factors needed to be considered while splitting the data. These additional criteria are discussed below and also summarized in Table 5.

First, rather than only splitting based on the number of waypoints, trajectories were grouped by the individual animal identifier as well as the year component of the row's timestamp. This ensured that trajectories were not split up arbitrarily, which could diminish the credibility of statistical evaluation. In my case, an individual's movement path within a calendar year was the actual "distinct entity". Cutting up these entities or shuffling them together would be counterproductive to the goal of path segmentation.

Grouping in this way also ensured that daily-downsampled trajectory sequences did not exceed 365 waypoints. Significantly imbalanced sequence lengths could also detract from model performance by introducing bias based on sequence length. Longer sequences are also more susceptible to the "vanishing gradient problem" [52], a challenge that is discussed in detail in Section 3.2.2. With a trajectory defined as an animal's path within one year (i.e. a sequence with a length less than or equal to 365), the final breakdown of trajectories per species is shown in Fig. 4.

After grouping trajectories by individual and year, the species distribution was taken into account. Due to the varying number of individuals in each species group, it was not directly possible to distribute them evenly. However, I at least ensured that each of the three output CSVs had multi-species representation (i.e. four or more species). The specific contents of species, waypoints, and trajectories per data subset are shown in Table 5 below. Looking at only the number of component trajectories, the training set has 78.7%, the validation set has 8.4%, and the testing set has 12.9% of the total paths.

A close inspection will reveal that *Anthropoides paradiseus* was only included in the training dataset. This was a deliberate decision since only two individuals were present in the data (see Table 2 above), their combined number of total waypoints only equaled 103, and their spatial distribution was an obvious outlier. Looking at the map in Fig. 1 above, it is clear that data for *Anthropoides paradiseus* only exists in South Africa rather than crossing larger regions of Eurasia. Future experiments might yield better results if *Anthropoides paradiseus* was excluded entirely, or if more records from that species were collected and divided amongst the data subsets. My research included them only in the training subset to test the robustness of training even in the presence of technically valid data outliers.

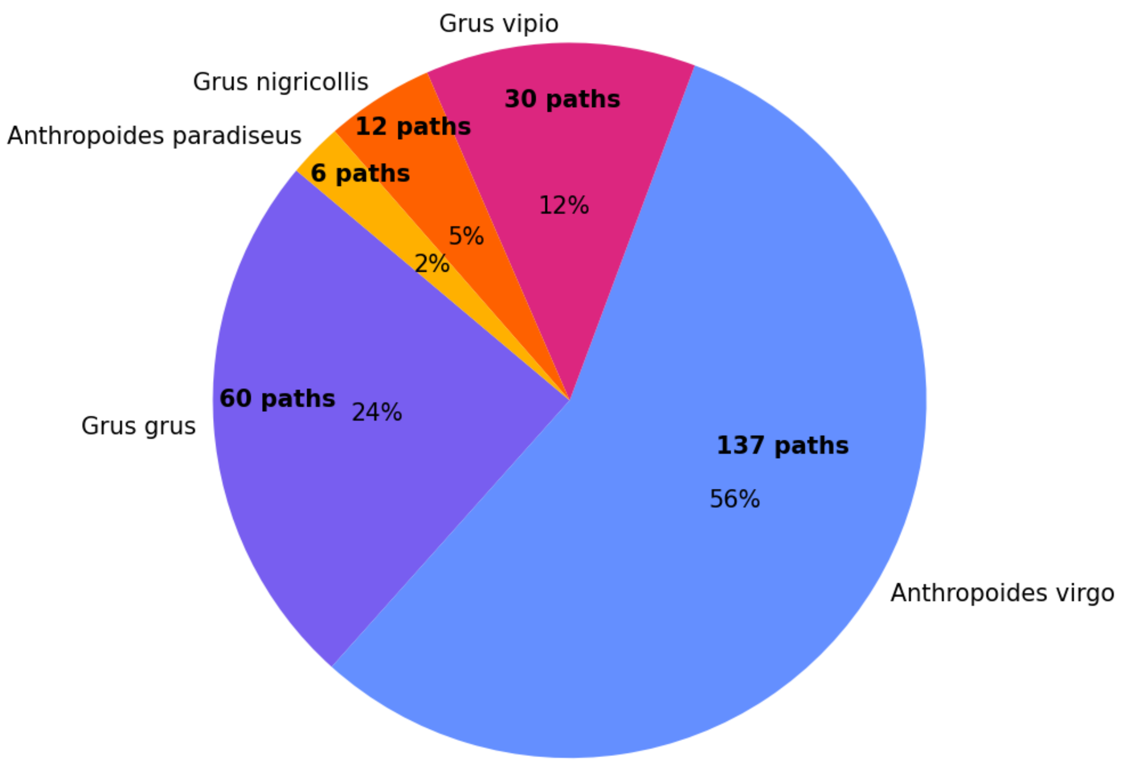


Fig. 4: Trajectory paths per species, where a path <= one year of one individual's activity

Data Subset	**Total Counts:** Trajectories *(Waypoints)*	**Species:** A. virgo	**Species:** G. grus	**Species:** G. vipio	**Species:** G. nigricollis	**Species:** A. paradiseus
Training	196 *(22,381)*	125 *(8,192)*	37 *(8,038)*	19 *(4,231)*	9 *(1,817)*	6 *(103)*
Validation	21 *(2,145)*	3 *(618)*	12 *(784)*	4 *(371)*	2 *(372)*	0 *(0)*
Testing	32 *(3,641)*	9 *(1,345)*	11 *(1,070)*	7 *(667)*	559 *(5)*	0 *(0)*

Table 5: Number of trajectories and waypoints per species, after data was split into three subsets

3.2 Machine Learning Model Workflow

3.2.1 Configuration and Hyperparameters

The starting point for the machine learning workflow implemented during this research was a configuration file where settings were defined before each training run. I used this as the basis for manual hyperparameter tuning, which is a critical part of improving a model's skill during training iterations [53].

The general groups of configuration options were: base model architecture, base optimizer, hyperparameters, number of training epochs, and various output settings. The complete list of configuration variables are described in Appendix 5. The implementation process is discussed in Supplement 2.

3.2.2 Model, Optimizer, and Scheduler Selection

The two base model architectures I used were a standard recurrent neural network (RNN) and a long short-term memory (LSTM) network [41] [54]. Both of these models are commonly used with time-series data since they are specialized for sequential input and effectively capture temporal patterns [41]. LSTMs were introduced to enhance the handling of extended sequences, offering this advantage over standard RNNs [54].

Long input sequences are prone to the “vanishing gradient problem”, a challenge that also manifests with “deep” (i.e. multi-layer) neural networks [52]. When training and validating a model on input data, researchers use a “loss function” to quantify the error (or “loss”) between the model’s classifications and ground-truth labels. They then use “gradient descent” to minimize the loss, attempting to find the steepest slope towards optimal classification. These error gradients are propagated backwards from the output layer to the input layer, passing through “activation functions” in a step called “backpropagation” [55]. Activation functions that clamp input values to a restricted range (e.g. between 0 and 1) can cause the gradients to progressively diminish as they propagate backwards through the network [52]. If left unchecked, this culminates in the gradual disappearance or “vanishing” of the gradient information [ibid]. This can slow or stall “model convergence” during training, impeding the attainment of optimal model parameters (i.e. the minimum loss) [ibid].

Instead of using an activation function that clamps between 0 and 1 (e.g. sigmoid), the default `PyTorch` RNN implementation uses the hyperbolic tangent (tanh) function [56]. Outputs of the tanh function span between -1 and 1, a wider range that enables more gradient information to be retained [57]. While this can partially mitigate the vanishing gradient problem, in practice the issue still persists in RNNs that use tanh – specifically with deeper neural networks or longer input sequences [ibid].

LSTMs were devised to address this RNN shortfall, introducing a “memory” component to enable learning from long-range inputs [54] (see Appendix 6 for an architectural diagram, and Section 3.2.3 for more details). Studies have shown that LSTMs often perform with higher accuracy than standard RNNs when receiving long sequential data [59][60]. As mentioned in Section 3.1, I limited sequence lengths by downsampling annual geolocation records to daily waypoints. With a maximum path length of 365, it was still worth evaluating whether the specialization of LSTMs would make a difference in the model performance. Therefore, my approach employed both standard RNNs as well as LSTMs to see which model was most effective.

After the model was initialized, I used it to create the “optimizer”. The optimizer is the training component that utilizes the gradients computed during backpropagation to adjust model parameters. Specifically, it changes the model’s weights and biases [61].

One popular optimization algorithm that I used was stochastic gradient descent (SGD). This technique randomly selects a portion of the training data on each iteration to update the model’s parameters [62]. In doing so, it introduces variability which prevents the model from converging to “local minima” [ibid]. Local minima are defined as points where the gradient slope becomes zero – representing a possible optimal solution – but when viewed in the context of the full parameter landscape (or “hyperspace”) are not at the *global* minimum [ibid]. Local minima can also be reached when the “learning rate” is too high, a hyperparameter defining the step size during gradient descent [63].

The other stochastic optimization algorithm used in my models was adaptive moment estimation (ADAM). The ADAM technique dynamically adapts the learning rate for each individual parameter, adjusting the step size based on historical gradient characteristics – namely, the first two moments [64]. The first moment is the moving mean of the gradient over past iterations, and the second moment is a measure of how much the gradient values varied [ibid]. By using this information, the ADAM optimizer can adapt to the parameter landscape on each iteration and efficiently navigate to converge on minima [ibid].

Another concept related to optimizers is the inclusion of “learning rate schedulers”. Schedulers can often improve the model’s accuracy by varying learning rates across different training epochs. Unlike ADAM’s adaptive learning rate which operates at the parameter level, a learning rate scheduler adjusts the learning rate *globally* based on observed model performance progress [63]. The scheduler that I used in my research tracked the loss value for each epoch. It globally reduced the learning rate by a specific factor when no improvement was seen for a certain number of epochs (i.e. when the model performance reached a “plateau”). This was useful for fine-tuning model performance gains towards the end of the training loop iterations [65].

3.2.3 Training and Validating the Model

The model training process consisted of: showing it examples of properly labeled movement path segments; letting it try to classify unlabeled path segments; determining when the model labels were wrong; and adjusting the model’s weights and biases so it could improve on the next iteration [61]. Validation scores for a separate input data subset were used to inform the hyperparameter tuning. The implementation of this process is described in detail in Supplement 3 and outlined as a block diagram in Fig. 5. Critical aspects are also described within this section, using the blocks from Fig. 5 as subsection headers.

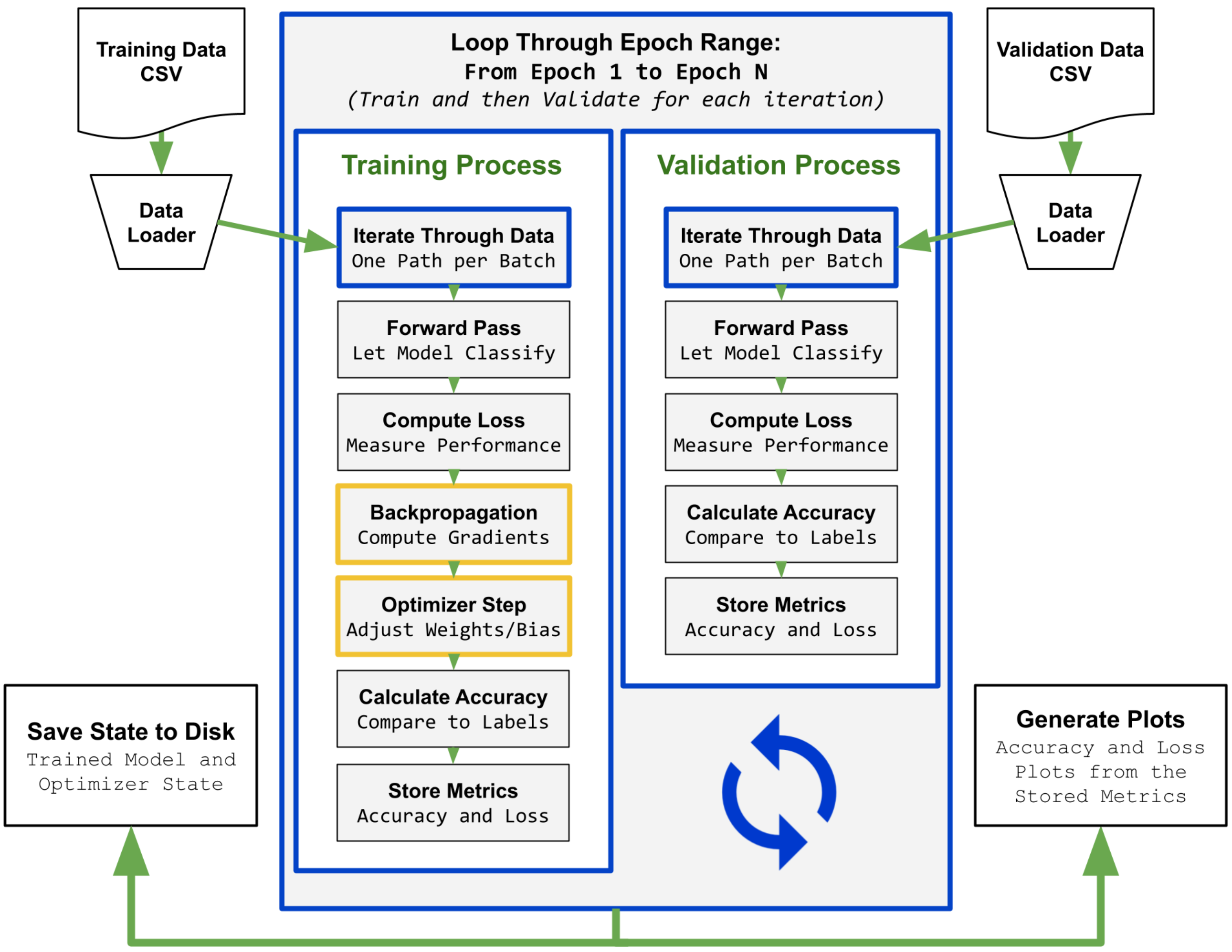


Fig. 5: Block diagram overview of the model training and validation workflow

Iterate Through Data

The training and validation processes both loaded data subsets, separated by the process described in Section 3.1.4. This ensured that the model learned characteristics from one dataset and successfully applied those insights on an entirely different set of data. “Overfitting” refers to when the model performs well on the training data, but poorly on other unseen sets of validation data. Separating subsets can help diagnose and identify overfitting, which in turn improves the model’s ability to generalize. This is crucial for the model’s performance when deployed in a real-world application, where its purpose is to classify sets of unlabeled data.

One strategy I employed for improving model generalization was shuffling training inputs during each new epoch. This reduced the chances of the model overfitting to a particular portion of the training set by preventing it from memorizing the order in which it received the input batches [66]. Another strategy is shuffling sequences inside of the batches themselves, but to limit the number of hyperparameter variables, I set a fixed batch size of 1. This meant that each batch contained a single sequence (i.e. path) with no more than 365 waypoints. In future evolutions of my research methodology, batches could include multiple path sequences. Shuffling those might also help the model to generalize further.

Forward Pass

The forward pass is when the model attempts to classify the inputs, passing them through the network of "hidden layers" to the output layer [55]. In an RNN, this involves stepping through the temporal sequences, with the first step initializing the hidden state [41]. In each following time step, the hidden state is updated with information from the current input and the previous hidden state [ibid]. In this way, historical information is preserved which allows the RNN to learn temporal patterns.

The same general process applies in an LSTM, with additional complexity in the form of three gates [54]. The "forget gate" determines information to discard or "forget" from the state of the previous step. The "input gate" determines which information should be added or updated in the current state. The "output gate" determines which of that information should be revealed to subsequent layers. By selectively controlling the flow of information through the model, these gates can help to address the vanishing gradient problem described in Section 3.2.2 [ibid]. A figure is provided in Appendix 6 with a visualization of these three gates.

Before passing input sequences through the model, I padded them to guarantee uniform length. This was required because not all trajectories contained the same total number of waypoints. Although the downsampling process ensured that paths had less than 365 waypoints, gaps in the original time-series data existed so not every day was captured. Once the padded sequences were packed and passed through the neural network, the next step was applying the loss function on the model outputs.

Compute Loss

When the goal of a model is classification – such as labeling paths with one of four classes – the loss function should produce probabilities that inputs belong to a certain class [67]. The cross-entropy loss function accomplishes this by combining the log-softmax function and the negative log-likelihood (NLL) loss [68]. Employing the cross-entropy loss function allowed my model to output the likelihood that input waypoints belonged to a specific segmentation class.

Backpropagation and Optimizer Step

Backpropagation and the optimizer step are highlighted with yellow in Fig. 5 and only take place during training (i.e. not during validation). These are the parts of the process described in Section 3.2.2, when gradients are computed to determine how model weights and biases should be adjusted.

In essence, this is the “learning” stage where model parameters are updated to minimize the loss between model-produced and ground-truth classifications. Backpropagation computes the gradients to find how each parameter affects the loss, and the optimizer uses these gradients to actually adjust the parameters. Together, these two steps enhance the model’s skill at classifying movement path segments on each iteration.

Calculate Accuracy and Store Metrics

Calculating the accuracy and loss of each epoch enables analysis of the training workflow. Once the loop in Fig. 5 was complete, two separate plots were generated: one with the training and validation accuracies, and one with the training and validation losses (Fig. 6). Inspecting these plots allowed me to confirm that the model improved across consecutive epochs – with accuracy moving towards higher values and loss moving towards lower values. In cases where the model performance plateaued, these plots also allowed me to see the effect of the learning rate scheduler described in Section 3.2.2. The model and optimizer states were saved after finishing the final epoch iteration.

3.2.4 Evaluating Model Performance

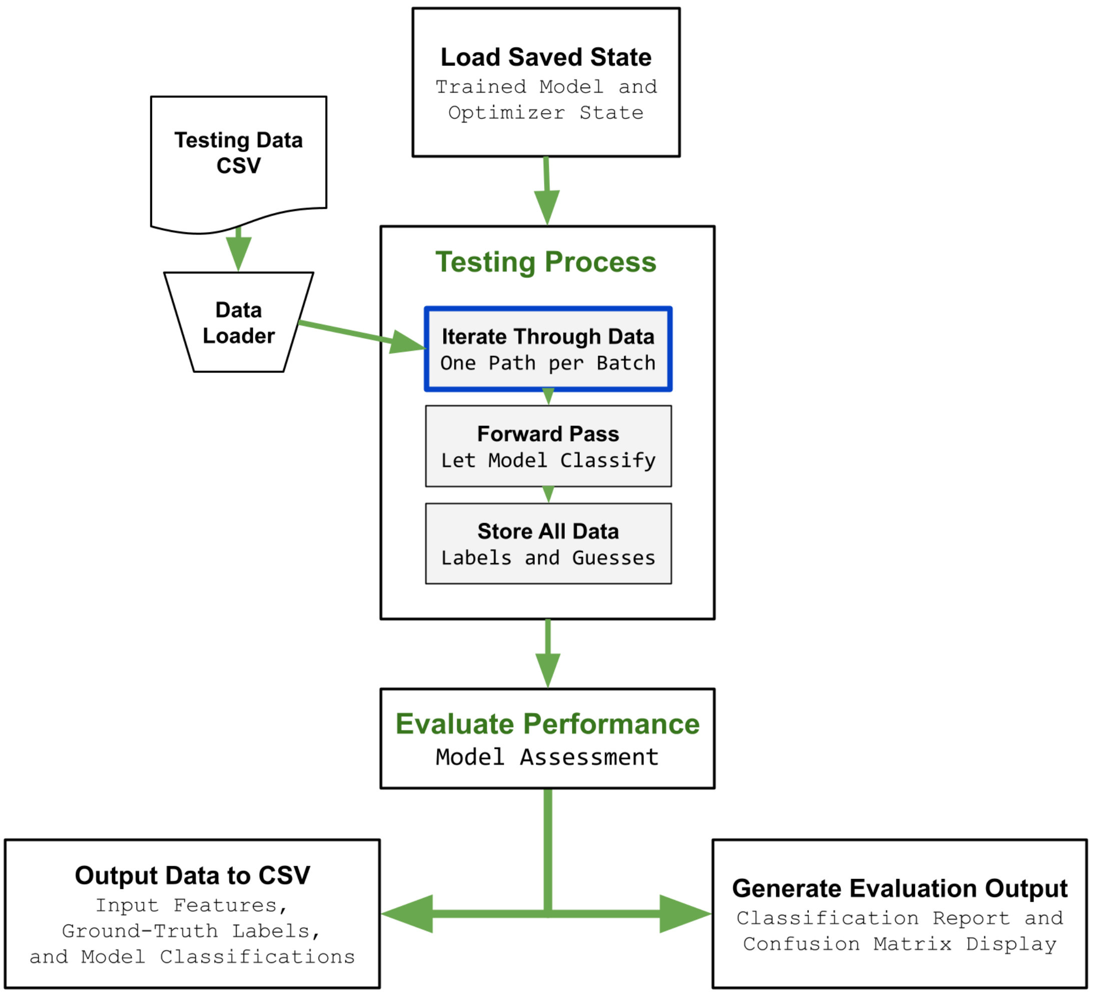


Fig. 6: Block diagram overview of the final model testing and performance evaluation

Performance Evaluation with "Unseen" Final Test Dataset

After loading the saved model and optimizer states, the final test subset was evaluated (see Fig. 6). Since the validation subset was used to inform hyperparameter tuning, the final test subset allowed for a truly independent assessment of performance. The model-produced labels were compared against the ground-truth labels and three main outputs were produced for reporting:

1. **Accuracy**: This single percentage value indicates the model's skill at succeeding in its task of classifying path segments. In order for a model-produced class to be "accurate", it must match the waypoint's ground-truth label [69].

2. **Classification Report**: This report (see Table 8) provides a more robust view of performance than a single accuracy score, which fails to capture possibilities such

as always predicting the same value or imbalanced distributions of classes [70]. Metrics included for each of the classes are outlined below. `tp` stands for “true positives”, `fp` stands for “false positives”, and `fn` stands for “false negatives”.

- **Precision:** $\frac{tp}{(tp + fp)}$

  This ratio measures the model’s ability to **not** label a negative sample as positive [ibid].

- **Recall:** $\frac{tp}{(tp + fn)}$

  This ratio measures the model’s ability to find all of the positive samples [ibid].

- **F1-score:** $\frac{tp}{tp + \frac{1}{2}(fp + fn)}$

  A weighted harmonic mean of the precision and recall, where the best case is a value of one and the worst case is a value of zero [ibid].

- **Support:** The number of occurrences of each ground-truth label in the input test data [ibid]. Therefore, the sum of all “support” values equals the total number of inputs.

- **Macro Average:** The unweighted mean for each metric, which does not account for class imbalance (i.e. varying “support” values) [ibid].

- **Weighted Average:** The weighted mean for each metric, which does account for class imbalance [ibid].

3. **Confusion Matrix:** This 4x4 matrix (see Fig. 8) shows the count of model predictions per class (X-axis) against the correct ground-truth labels (Y-axis). A perfect model would only show values in the diagonal line from top-left to bottom-right of the matrix. Values in any cell outside of this diagonal are indicative of the model making an incorrect classification. Since each row corresponds to a specific ground-truth label on the Y-axis, the sum of all horizontal cells in a row equals the relative “support” value for that label [71].

These evaluation metrics are all included in Section 4. Together, they offer a comprehensive view on the skill of the model against previously unseen data. They show the model’s ability to classify movement paths, and also confirm that the model didn’t overfit to the training set.

To produce my final results, I used a high-performance computer (HPC) system. The process for training and evaluating on an HPC is described in detail in Supplement 4.

4 MODEL RESULTS

4.1 Results and Discussion

Top Results Before Adding Intra-day Features

Model	Optimizer	Num Layers	Hidden Size	Dropout	Accuracy: % correct labels
RNN	**ADAM**	**2**	**32**	**0.2**	**81.49%**
RNN	ADAM	2	64	0.3	81.05%
RNN	ADAM	2	32	0.3	80.91%
RNN	ADAM	2	32	0.1	80.83%
RNN	ADAM	3	128	0.5	80.78%
RNN	SGD	2	64	0.4	80.34%
LSTM	ADAM	2	32	0.1	80.01%
LSTM	ADAM	4	128	0.2	79.87%

Table 6: Best model performance evaluations without including intra-day features

Top Results After Adding Intra-day Features

Model	Optimizer	Num Layers	Hidden Size	Dropout	Accuracy: % correct labels
RNN	**ADAM**	**3**	**128**	**0.3**	**83.27%**
RNN	ADAM	2	128	0.2	82.20%
RNN	ADAM	2	64	0.2	82.09%
RNN	ADAM	2	32	0.2	82.01%
LSTM	ADAM	3	128	0.3	80.58%
RNN	ADAM	4	128	0.3	80.36%
RNN	ADAM	4	128	0.2	80.25%
RNN	ADAM	4	128	0.2	80.12%

Table 7: Best model performance evaluations, with top result in green. Including all intra-day features: (mean daily distance, mean daily velocity, meany daily bearing, mean daily turn angle)

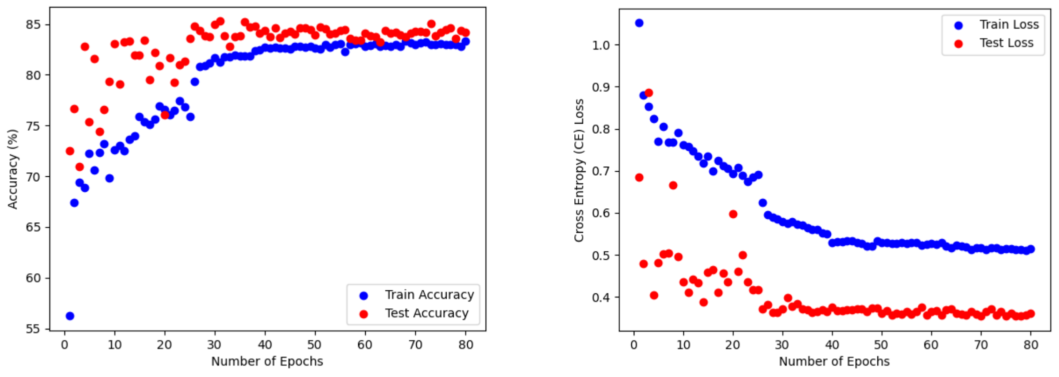
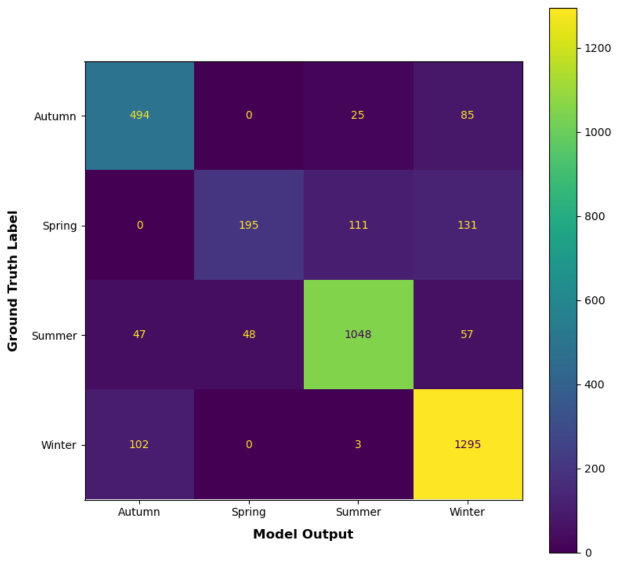


Fig. 7: Final accuracy and loss plots for all training epochs (corresponds to best-performing model run)



Ground Truth Label \ Model Output	Autumn	Spring	Summer	Winter
Autumn	494	0	25	85
Spring	0	195	111	131
Summer	47	48	1048	57
Winter	102	0	3	1295

Fig. 8: Final confusion matrix display (corresponds to best-performing model run)

	precision	**recall**	**f1-score**	**support**
Autumn:	0.77	0.82	0.79	604
Spring:	0.80	0.45	0.57	437
Winter:	0.88	0.87	0.88	1200
Summer:	0.83	0.93	0.87	1400
accuracy:			0.83	3641
macro avg:	0.82	0.77	0.78	3641
weighted avg:	0.83	0.83	0.83	3641

Table 8: Final classification report (corresponds to best-performing model run)

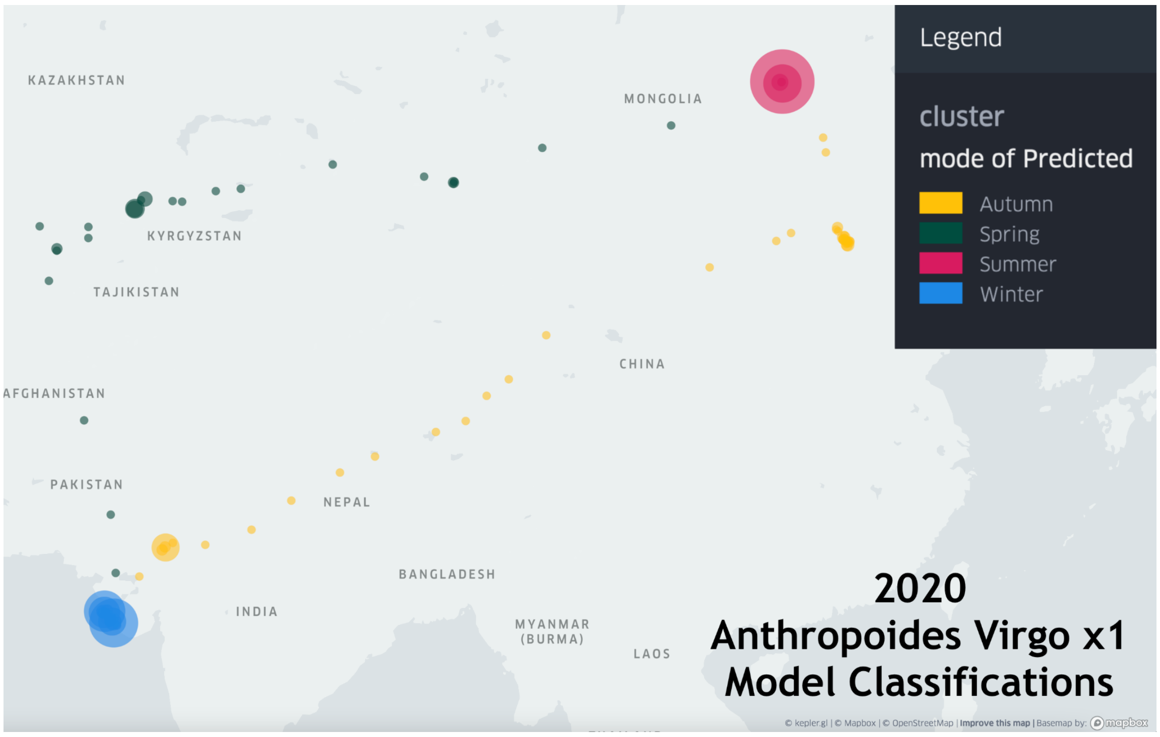


Fig. 9: Model classifications for daily-downsampled waypoints from one distinct path (filtered by year) for one Anthropoides virgo *individual; with spatial clustering to set circle size based on waypoint density*

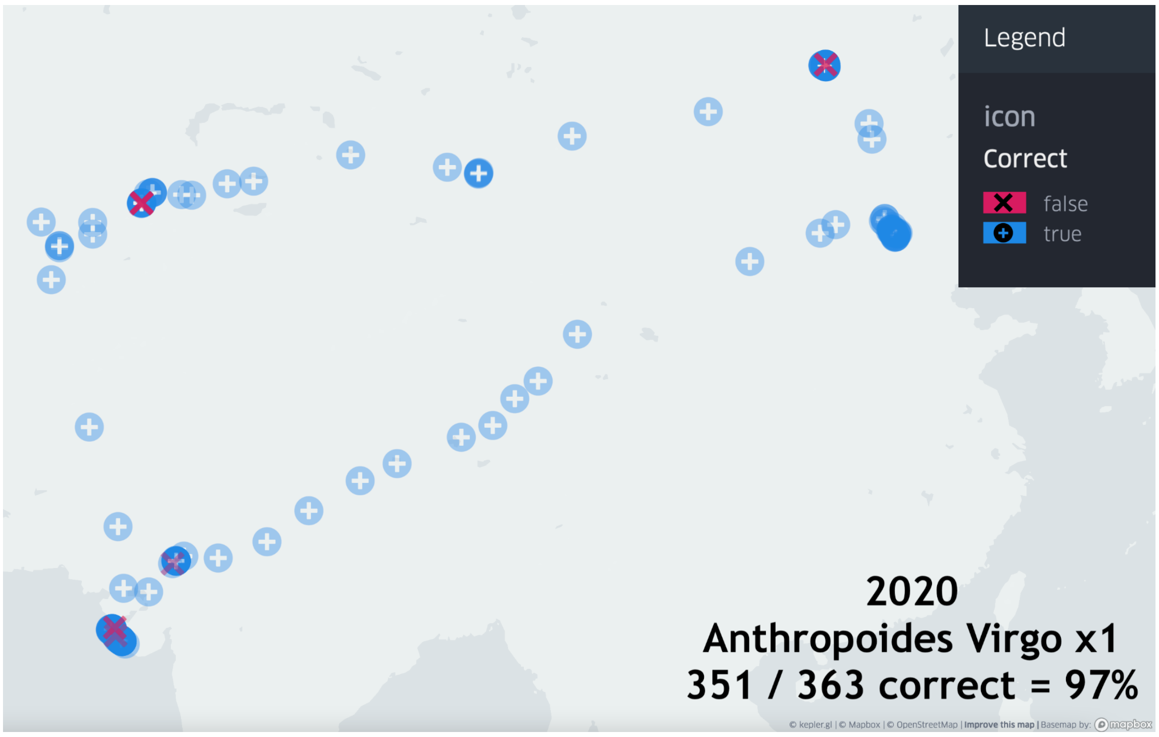

Legend
icon
Correct
false
true
2020
Anthropoides Virgo x1
351 / 363 correct = 97%
© kepler.gl | © Mapbox | © OpenStreetMap | Improve this map | Basemap by: mapbox

Fig. 10: Correct (true) and incorrect (false) model classifications for daily-downsampled waypoints from one distinct path (filtered by year) for one Anthropoides virgo *individual*

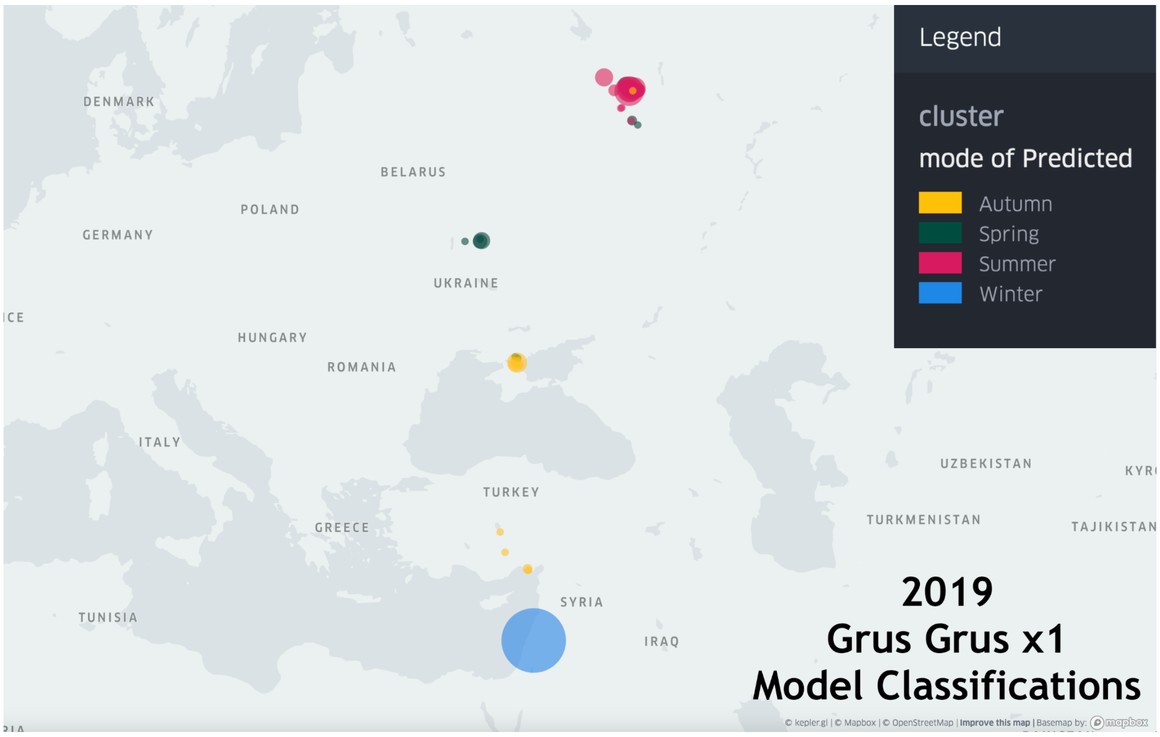


Fig. 11: Model classifications for daily-downsampled waypoints from one distinct path (filtered by year) for one Grus grus *individual; with spatial clustering to set circle size based on waypoint density*

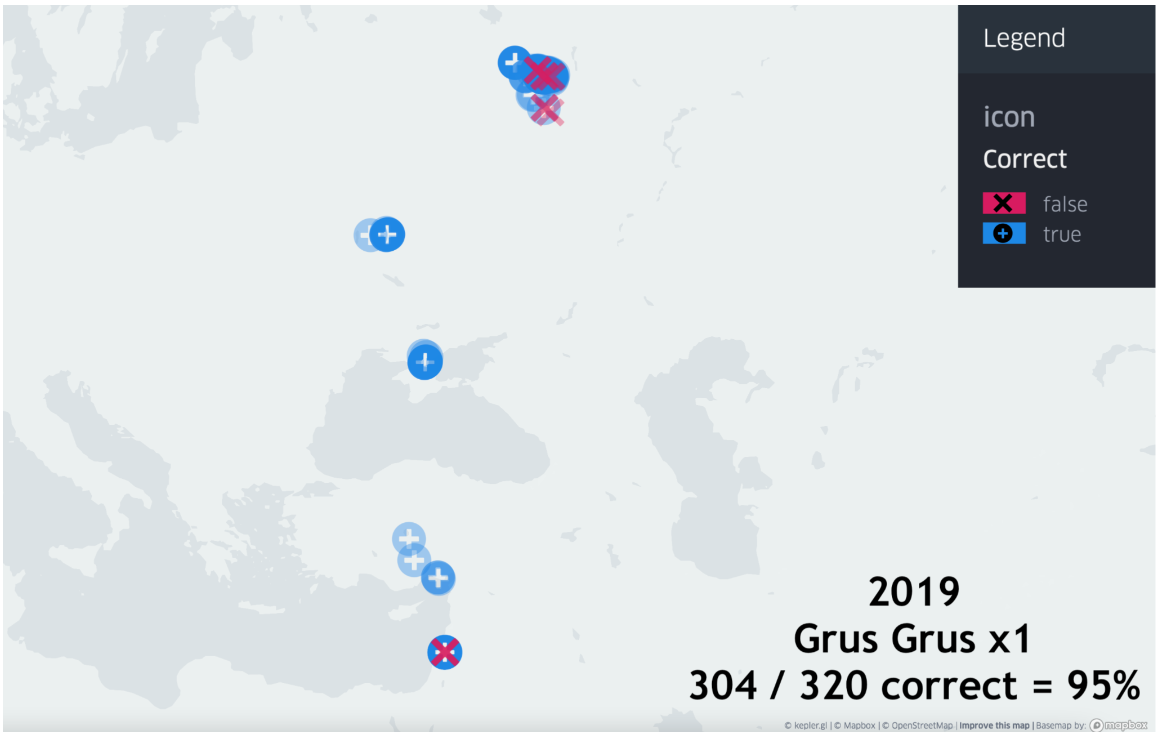

Legend
icon
Correct
✗ false
⊕ true
2019
Grus Grus x1
304 / 320 correct = 95%
© kepler.gl | © Mapbox | © OpenStreetMap | Improve this map | Basemap by: mapbox

Fig. 12: Correct (true) and incorrect (false) model classifications for daily-downsampled waypoints from one distinct path (filtered by year) for one Grus grus *individual*

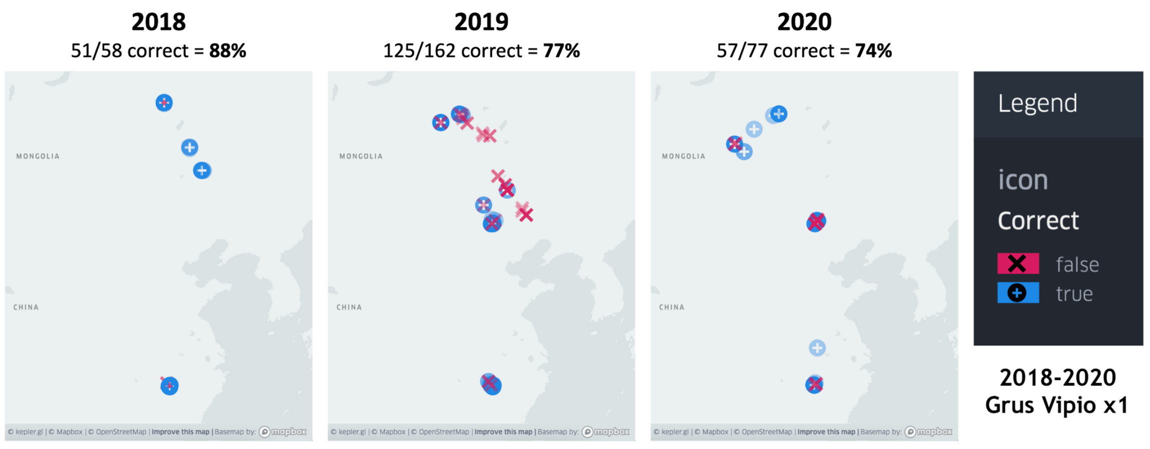


Fig. 13: Correct (true) and incorrect (false) model classifications for daily-downsampled waypoints from three distinct paths (filtered by year) for the same Grus vipio *individual*

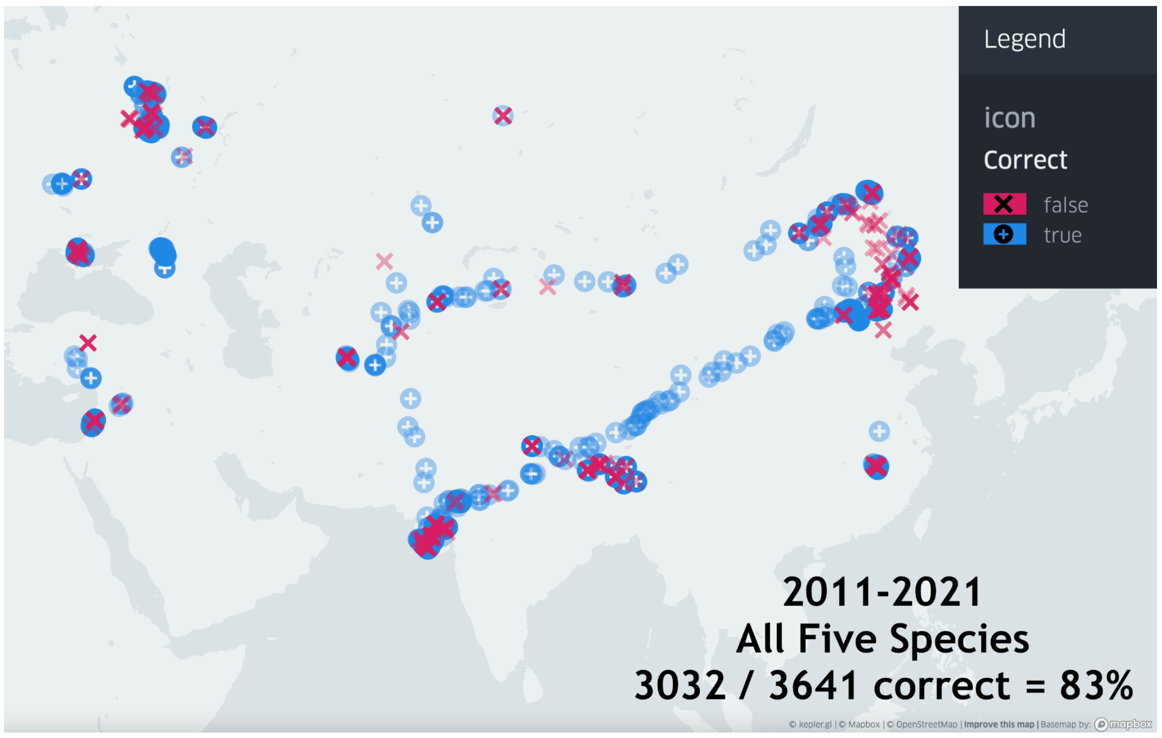


Fig. 14: Correct (true) and incorrect (false) model classifications for daily-downsampled waypoints from all paths for all five species in the final test data subset

Table 6 shows the accuracies and relevant hyperparameters for eight different model configurations, before including any daily mean movement values. Even without intra-day features, these models achieved accuracies between 79.87% and 81.49%. Table 7 shows the results for the top eight model configurations after daily mean features were included as inputs. After adding intra-day features, these models achieved accuracies between 80.12% and 83.27%. Fig. 7 shows the accuracy and loss plots produced by the most successful model's training process (i.e. the model with 83.27% accuracy). Fig. 8 and Table 8 display the confusion matrix and classification report for that same model (with terms defined in Section 3.2.4). Figures 9 through 14 all contain results from the best-performing model, and were all created with `kepler.gl` [46].

Fig. 9 visualizes a single trajectory from one *Anthropoides virgo* individual – the species with the highest percentage of paths in the daily-downsampled dataset (see Fig. 4). The map visualization uses spatial clustering to increase the size of the circle icons in areas where the density of waypoints is higher [72]. The colors of the clusters are based on the mode of the model's classifications for the waypoints in each cluster. With 363 waypoints represented, the figure depicts one calendar year. Winter and summer range residencies, along with vernal and autumnal migrations, are clearly demarcated by the model's segmentations. Fig. 10 removes the spatial clustering, but visualizes the same trajectory. It styles the waypoints based on whether the model classification was correct. 351 out of the 363 waypoints were classified correctly, resulting in a single trajectory accuracy of 97%.

Fig. 11 employs the same visualization process as Fig. 9, but shows a single trajectory from one *Grus grus* individual. The *Grus grus* species had the second highest percentage of paths in the daily-downsampled dataset (see Fig. 4). Fig. 11 also depicts the majority of one year, with 320 waypoints. Although the four classes are also demarcated by the model's segmentations, there are some *"Autumn"* and *"Spring"* waypoints that appear in the mostly *"Summer"* region. Fig. 12 – using the same visualization process as Fig. 10 – shows that the model did produce incorrect classifications in that specific region. However, 304 out of 320 waypoints were still classified correctly, resulting in a single trajectory accuracy of 95%.

Fig. 13 uses the same visualization process as Fig. 10 and 12, and shows three consecutive annual paths for one *Grus vipio* individual. The *Grus vipio* species had the third highest percentage of paths in the daily-downsampled dataset (see Fig. 4). With 30 trajectories, *Grus vipio* represented half of the amount of *Grus grus* paths and only 18.4% of the paths available for *Anthropoides virgo.* Looking at the three yearly paths for one *Grus vipio* individual in Fig. 13, it is clear that the model performed slightly worse. However, for the 2018 trajectory, the model still correctly classified 51 of 58 waypoints. This resulted in a single trajectory accuracy of 88%. For the 2019 path, the model correctly classified 125 of 162 waypoints with an accuracy of 77%. For the 2020 path, the model correctly classified 57 of 77 waypoints with an accuracy of 74%. It is notable that none of these yearly trajectories exceeded 162 waypoints, so even the path with the most waypoints still represented less than half of 365. Lacking so much of the yearly time-series might help to explain the reduced performance.

Fig. 14 visualizes all trajectories in the final test subset. 3032 out of 3641 waypoints were correctly classified by the model, resulting in an overall performance accuracy of 83%. Certain areas appear to have higher occurrences of incorrect classifications, such as the general region of Inner Mongolia (next to the legend in Fig. 14). Many of those incorrect waypoints belong to a single *Grus vipio* trajectory (a different individual than the one from Fig. 13), with 168 of 247 waypoints correctly classified and a single trajectory accuracy of 68%.

The plots generated during the training process show the history of how the model learned. Plots of accuracy and loss values for sequential training and validation steps illustrate that the model improved while iterating through the epoch range (Fig. 7). This is shown by the fact that accuracies increased and losses decreased.

The confusion matrix visualization produced for the best-performing model shows the spread of the model's final test set classifications (Fig. 8). It demonstrates that the model did not simply choose the most common class, but rather uncovered data patterns that it used to achieve its learning goal. Although the confusion matrix is not a perfect diagonal from top-left to bottom-right, the higher quantity of records in those diagonal cells confirm that model classifications generally matched the ground-truth labels.

One interesting observation is that the "memory" component of the LSTM – which allows the model to maintain more information regarding previous inputs – did not significantly change the results when compared to the basic RNN. Just two of the 16 configurations from Tables 6 and 7 used an LSTM base model that exceeded 80% (one in each table), and the only model from this group with an accuracy below 80% was also an LSTM.

This could possibly indicate that determining animal movement transitions (i.e. from one behavioral state to another) is not very dependent on the long-term movement history. Including intra-day features boosted results by a couple of percentage points, but results still exceeded 80% without them. This observation aligns with reported results from methods in Table 1 – from the stepwise approach of BCPA to the limited memory of HMMs – but additional dedicated research is needed to validate the claim. Although my models were trained and tested on five separate avian species, this intuition regarding the amount of required movement history may not apply beyond that scope.

Model accuracies and F1-scores over 80% were achieved in 15 out of the 16 configurations shown in Tables 6 and 7. When analyzing the classification report of the best-performing model run (Table 8), I did observe that the recall score of the *"Spring"* class was less than the others – showing that the model was not as skillful at identifying positive *"Spring"* values. Since the *"Spring"* and *"Autumn"* classes represented less records than the other two classes of *“Winter”* and *“Summer”*, increasing the total dataset size might improve this metric in future experiments.

Extensive ablation studies were performed to empirically assess different hyperparameters. Tables 6 and 7 only include the most relevant hyperparameter variations, but the other investigated values are all described in Appendix 5.

One important hyperparameter was neural network depth, meaning the number of hidden layers specified for the model. Another was the individual layer dimensionality – or “size”– defined for each hidden layer [55]. In my research, increasing the number of layers and hidden size beyond the values in Tables 6 and 7 did not yield better results.

A third hyperparameter included in those tables was the “dropout” percentage, which did affect performance. Dropout is a regularization technique that randomly deactivates some of the model’s neurons during each training iteration [73]. This introduction of randomness can help to mitigate overfitting because it prevents the model from building strong dependencies on specific neurons [ibid].

Hyperparameters such as weight decay and SGD momentum had minimal effect on model performance. However, the *type* of optimizer was evidently important, with ADAM largely outperforming SGD variations.

The initial learning rate was investigated too, but was set to 0.001 for all of the top results. I also varied and analyzed the number of training epochs, with a maximum of 80 for the best-performing runs. After 80 epochs, the performances tended to stop improving. The scheduler employed to reduce the learning rate when model skill plateaued did result in slight performance gains for later epochs.

5 FUTURE WORK

My research questions specifically focused on assessing the accuracy of recurrent neural networks *without* the inclusion of environmental features. Therefore, throughout the course of my research, I excluded any such model inputs. However, I did define a process to retrieve this data. I also began reviewing studies to guide additional feature selection. Now that I have set an accuracy benchmark with time and movement features, future efforts can focus on testing the model with environmental data.

5.1 Integrating Environmental Features

After concluding my research as described above, I implemented a process to download environmental data. This process makes requests using latitude, longitude, and timestamp values from animal geolocation records. The CDS API is provided through the European Union's “Copernicus” program, in partnership with the European Centre for Medium-Range Weather Forecasts (ECMWF) [74]. Available data includes both Earth Observation and climatological records, with variables such as: temperature, relative humidity, precipitation, wind speed and direction, solar radiation, soil moisture content, soil type, high and low vegetation cover, type of vegetation, and many other atmospheric and Earth-surface metrics [ibid].

Using this data within my model requires merging the downloaded fields into the main dataset by joining them on geocoordinate and timestamp values. Then they can be normalized, split into subsets, and added to the model as auxiliary features. However, special attention must be given to data resolution. Many of the variables have known effects on fauna – such as wind-mediated movements during transoceanic flights – but the coarse resolution of the data in space and time can possibly diminish insights [75].

Beyond CDS, there are other datasets that could also be included. Land surface phenology can help to identify biodiversity changes induced by large spatial scale perturbations [76]. The Normalized Difference Vegetation Index (NDVI) is useful in predicting fauna distribution, population, and life history states [77]. Again, procedures to increase the resolution of data are often required. Since Earth Observation satellite missions trade between spatial and temporal resolution [78], choosing appropriate datasets should be done on a case-by-case basis.

Along with data on the natural world, human activity could be considered. Nocturnally migrating birds are susceptible to deviations caused by artificial light in urban areas [79][80]. Nighttime radiation is provided by NOAA/EOG VIIRS data [81], which is further corrected in NASA's Black Marble project [82]. Other areas of active research investigate noise pollution and how it affects both fauna population density and movement patterns [83][84]. The Global Roads Open Access Data Set is one proxy for human-created noise offered in NASA’s Earthdata catalog [85].

Finally, some caveats should be considered when possibly including additional features. Expanding the number of inputs will increase the model’s complexity. Too many specialized features can reduce the model’s ability to generalize [86]. In the worst case scenario, irrelevant information can lead to decreased performance [87]. A useful guideline is to independently verify any expected signal. Otherwise, the model might identify a pattern which doesn't contribute to the learning objective.

6 CONCLUSION

This manuscript explored the application of RNN models to the task of classifying satellite-collected animal movement data. Regarding my first research question, a maximum F1-score of 83.27% was achieved. Regarding my second research question, 15 out of the 16 top-performing models achieved accuracies over 80% – none of which required any environmental features. When including intra-day features of daily mean movement values, the top eight accuracies were all between 80.12% and 83.27%. When excluding intra-day values – minimizing the feature set even more – the top eight accuracies were still between 79.87% and 81.49%.

The F1-scores show that both RNNs and LSTMs were applicable, though they do leave room for future improvements – many of which have been discussed. Now that an accuracy benchmark is set without including habitat features, these can be incorporated to boost performance metrics. I cited several sources for auxiliary environmental inputs, some that are already widely used and others that require additional study.

Benefits of my approach are its suitability for larger datasets and the degree of control offered by the `PyTorch` framework. Another optimization that this method supports is algorithmic hyperparameter tuning. One technique is stepping through a grid of hyperparameter combinations to determine which yields the best performance. A grid search methodology still enables fine control of the system without relying on more opaque and off-the-shelf enhancements.

In this manuscript, I have outlined my method and steps for its further improvement. Researchers now have this tool as a resource to fine-tune for their specific needs. Future movement ecology studies will require such tools to analyze the rapidly growing volumes of data from space-based tracking systems. Combined with efforts from IoA and other animal tracking groups, these methods can accelerate global biosphere research.

ACKNOWLEDGEMENTS

I am deeply grateful for all of the support received from my team of thesis advisors:

Dr. Scott Yanco – who recently moved from Yale BCG to the University of Michigan – was instrumental in defining the original research goals. His fluent knowledge of movement ecology, avian behavior, and analytical methods was fundamental to the success of this project.

Dr. Benjamin Kellenberger – from Yale BCG – offered extensive technical guidance regarding recurrent neural networks, and machine learning best practices in general. Several times, the project might have hit a dead end if it weren't for his wealth of expertise on these subjects.

Dr. Jan Thoemel – from the University of Luxembourg – has been a pillar of support for multiple years. From lectures on CubeSats and orbital mechanics to feedback on scientific writing, his wisdom has also been invaluable.

Along with my official thesis advisors, whose direct contributions can not be overstated, I am also indebted to a host of others involved along the way:

I am grateful to Dr. Ryan Pavlick of the NASA Jet Propulsion Laboratory (JPL) for bringing me into the Internet of Animals community and their research. After taking the time to read my internship application to JPL, he reached out directly to help me find the right path. In our first meeting, he introduced me to the IoA initiative, and convinced me of its importance with fascinating case studies. The climate change feedback loop involving North American beavers was one such study that he brought to my attention.

Later meetings with others from the IoA community – such as Dr. Alex Killion (Managing Director at Yale BGC) and Dr. Antonio Ferraz (Scientist at NASA JPL) – continued to inspire as I learned even more. They further assured me that many great minds are contributing to the IoA initiative.

My sincere thanks also goes to Dr. Lia Vogas, Dr. Cody Folta, and Dr. Scott Yanco (all associates of Yale BCG) for their time in producing the crane migration labels. The labeling was done through an app developed by Dr. Fabiola Iannarilli. Without the combined work from this team, there would have been no ground-truth for the model. Their time and efforts were a critical part of this project from the start.

Prior to meeting anyone from the IoA group, I spoke with Dr. Alex Gardner (Program Manager / Scientist at NASA JPL) as well as Dr. Dana Yoerger (Senior Scientist at Woods Hole Oceanographic Institution), both of whom were very supportive and generous with their time. My conversations with each of them helped to hone my research interests, and I learned a great deal from them that guided me on my way. If it were possible to do three MSc theses simultaneously, I would have loved to work with both of their research teams too.

Finally, I'd like to acknowledge the animals, without whom none of this would be possible. Their contributions are important reminders that life in all forms is a precious gift. It is my sincere hope and desire that this research and future iterations assist in reducing suffering for *all* of our planet's sentient beings.

REFERENCES

1. NASA ESDS Program. *"Satellite Needs Working Group Solutions: In Implementation: Animal Tracking (Internet of Animals)"*. https://www.earthdata.nasa.gov/esds/impact/snwg/solutions. Accessed 10 July 2023.
2. Yale University Center for Biodiversity and Global Change. *"Internet of Animals Symposium"*. https://bgc.yale.edu/internet-animals-symposium. 31 January 2023.
3. Ruth Y. Oliver, Carsten Meyer, Ajay Ranipeta, Kevin Winner, Walter Jetz. *"Global and national trends, gaps, and opportunities in documenting and monitoring species distributions"*. https://doi.org/10.1371/journal.pbio.3001336. 12 August 2021.
4. Roland Kays , Margaret C. Crofoot, Walter Jetz, Martin Wikelski. *"Terrestrial animal tracking as an eye on life and planet"*. https://doi.org/10.1126/science.aaa2478. 12 June 2015.

5. American Institute of Biological Sciences, Washington D.C. Bioinstrumentation Advisory Council. *"Some Prospects for Using Communications Satellites in Wild Animal Research"*. https://apps.dtic.mil/sti/citations/tr/AD0647291. 1 July 1966.
6. US Fish and Wildlife Service. *"Satellite Telemetry: A New Tool for Wildlife Research and Management".* https://apps.dtic.mil/sti/citations/ADA322683. 1 January 1988.
7. Movebank. *Homepage.* https://www.movebank.org/cms/movebank-main. Accessed 10 July 2023.
8. Greg Shirah, NASA Scientific Visualization Studio. *"Ecological insights from three decades of animal movement tracking across a changing Arctic".* https://svs.gsfc.nasa.gov/4877. 5 April 2021.
9. Ben Carlson. *"mosey_db: Local implementation of Movebank database"*. https://github.com/benscarlson/mosey_db. Initial commit: October 2020.
10. Michael Perras, Silke Nebel. *"Satellite Telemetry and its Impact on the Study of Animal Migration".* Nature Education Knowledge. https://www.nature.com/scitable/knowledge/library/satellite-telemetry-and-its-impact-on-the-94842487/. 2012.
11. Universität Konstanz. *"ICARUS – Tracking animals from space".* https://www.campus.uni-konstanz.de/en/science/icarus-tracking-animals-from-space#slide-5. Accessed 10 July 2023.
12. ICARUS Consortium (via European Space Agency's eoPortal). *"ICARUS Ground Segment".* https://www.eoportal.org/other-space-activities/iss-icarus. 30 June 2023.
13. Max Planck Institute for Animal Behavior. *"End of the cooperation".* https://www.icarus.mpg.de/en. 10 October 2022.
14. Walter Jetz, Grigori Tertitski, Roland Kays, Uschi Mueller, Martin Wikelski, et al. *"Biological Earth observation with animal sensors".* https://doi.org/10.1016/j.tree.2021.11.011. April 2022.
15. Max Planck - Yale Center for Biodiversity Movement and Global Change (MPYC). *"Digital Museum of Animal Lives: Following the lives of mobile organisms"*. https://animallives.org/. Accessed 11 July 2023.
16. Max Planck Institute for Animal Behavior. *"ICARUS flies faster, further".* https://www.ab.mpg.de/542769/news_publication_20614880_transferred?c=3273. 11 July 2023.
17. Cynthia A. Evans and Julie A. Robinson, Earth Sciences and Image Analysis, NASA Johnson Space Center. *"Space Station Orbit Tutorial".* https://eol.jsc.nasa.gov/Tools/orbitTutorial.htm. Accessed 13 July 2023.
18. University of Warwick WUSAT Programme Team. *"WUSAT-3 CubeSat Project".* https://warwick.ac.uk/fac/sci/eng/meng/wusat/projects/wusat-3/. Accessed 12 July 2023.
19. Frank Flechtner, et al. *"Realization of a Satellite Mission GRACE-I for Parallel Observation of Changing Global Water Resources and Biodiversity".* American Geophysical Union. https://ui.adsabs.harvard.edu/abs/2020AGUFMG020...07F/abstract. December 2020.
20. Ding Li Yong, Yang Liu, Bing Wen Low, Carmela P. Española, Chang-Yong Choi, Kazuto Kawakami. *"Migratory songbirds in the East Asian-Australasian Flyway: a review from a conservation perspective".* Bird Conservation International. https://doi.org/10.1017/S0959270914000276. 10 February 2015.

21. A.J. Gallagher, J.W. Brownscombe, N.A. Alsudairy, et al. *“Tiger sharks support the characterization of the world’s largest seagrass ecosystem”.* Nat Commun 13, 6328. https://doi.org/10.1038/s41467-022-33926-1. 1 November 2022.
22. Paul O'Donoghue, Christian Rutz. *“Real-time anti-poaching tags could help prevent imminent species extinctions”.* The Journal of Applied Ecology. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4949716/. February 2016.
23. Michele B. Parsons, Thomas R. Gillespie, Elizabeth V. Lonsdorf, Dominic Travis, Iddi Lipende, Baraka Gilagiza, Shadrack Kamenya, Lilian Pintea, Gonzalo M. Vazquez-Prokopec. *“Global Positioning System Data-Loggers: A Tool to Quantify Fine-Scale Movement of Domestic Animals to Evaluate Potential for Zoonotic Transmission to an Endangered Wildlife Population”.* https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0110984. 3 November 2014.
24. Martin Wikelski, Uschi Mueller, Paola Scocco, Andrea Catorci, Lev V. Desinov, Mikhail Y. Belyaev, Daniel Keim, Winfried Pohlmeier, Gerhard Fechteler, P. Martin Mai. *“Potential short-term earthquake forecasting by farm animal monitoring”.* https://doi.org/10.1111/eth.13078. 3 July 2020.
25. Marlee A. Tucker, Katrin Böhning-Gaese, William F. Fagan, John M. Fryxell, Bram Van Moorter, Susan C. Alberts, Abdullahi H. Ali, et al. *“Moving in the Anthropocene: Global reductions in terrestrial mammalian movements”*. https://www.science.org/doi/10.1126/science.aam9712. 26 January 2018.
26. Jakob Lundberg, Fredrik Moberg. *“Mobile Link Organisms and Ecosystem Functioning: Implications for Ecosystem Resilience and Management”.* https://link.springer.com/article/10.1007/s10021-002-0150-4. January 2003.
27. Marie A. Tremblay, Colleen C. St. Clair. *“Factors affecting the permeability of transportation and riparian corridors to the movements of songbirds in an urban landscape”.* https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/j.1365-2664.2009.01717.x. 26 November 2009.
28. Amanda J. Zellmer, Barbara S. Goto. *“Urban wildlife corridors: Building bridges for wildlife and people”.* https://www.frontiersin.org/articles/10.3389/frsc.2022.954089/full. 14 October 2022.
29. Siân E. Green, Zeke Davidson, Timothy Kaaria, C. Patrick Doncaster. *“Do wildlife corridors link or extend habitat? Insights from elephant use of a Kenyan wildlife corridor”*. https://onlinelibrary.wiley.com/doi/abs/10.1111/aje.12541. 29 November 2018.
30. Victoria Rawn Wyllie de Echeverria, Thomas F. Thornton. *“Using traditional ecological knowledge to understand and adapt to climate and biodiversity change on the Pacific coast of North America”.* https://link.springer.com/article/10.1007/s13280-019-01218-6. 9 October 2019.
31. Matthew J. Kauffman, Francesca Cagnacci, Simon Chamaillé-Jammes, Mark Hebblewhite, J. Grant C. Hopcraft, Jerod A. Merkle, Thomas Mueller, Atle Mysterud, Wibke Peters, et al. *“Mapping out a future for ungulate migrations”.* https://www.science.org/doi/abs/10.1126/science.abf0998. 7 May 2021.
32. Benjamin M. Jones, Ken D. Tape, Jason A. Clark, Allen C. Bondurant, Melissa K. Ward Jones, Benjamin V. Gaglioti, Clayton D. Elder, Chandi Witharana, Charles E. Miller.

"Multi-Dimensional Remote Sensing Analysis Documents Beaver-Induced Permafrost Degradation, Seward Peninsula, Alaska". https://www.mdpi.com/2072-4292/13/23/4863. 30 November 2021.

33. Toby A. Patterson, Marinelle Basson, Mark V. Bravington, John S. Gunn. *"Classifying movement behaviour in relation to environmental conditions using hidden Markov models".* https://pubmed.ncbi.nlm.nih.gov/19563470/. 26 June 2009.
34. Ian D. Jonsen, Ransom A. Myers, Michael C. James. *"Identifying leatherback turtle foraging behaviour from satellite telemetry using a switching state-space model".* http://dx.doi.org/10.3354/meps337255. May 2007.
35. Rémi Patin, Marie-Pierre Etienne, Emilie Lebarbier, Simon Chamaillé-Jammes, Simon Benhamou. *"Identifying stationary phases in multivariate time series for highlighting behavioural modes and home range settlements".* https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/1365-2656.13105. 20 September 2019.
36. David W. Wolfson, David E. Andersen, John R. Fieberg. *"Using piecewise regression to identify biological phenomena in biotelemetry datasets".* https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/1365-2656.13779. 19 July 2022.
37. Eliezer Gurarie, Russel D. Andrews, Kristin L. Laidre. *"A novel method for identifying behavioural changes in animal movement data".* https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1461-0248.2009.01293.x. 6 April 2009.
38. 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, Tanya Berger-Wolf. *"Perspectives in machine learning for wildlife conservation".* https://www.nature.com/articles/s41467-022-27980-y. 9 February 2022.
39. Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, Jonathan Huang. *"Context R-CNN: Long Term Temporal Context for Per-Camera Object Detection".* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). https://openaccess.thecvf.com/content_CVPR_2020/html/Beery_Context_R-CNN_Long_Term_Temporal_Context_for_Per-Camera_Object_Detection_CVPR_2020_paper.html. June 2020.
40. Cory Overton, Michael Casazza, Joseph Bretz, Fiona McDuie, Elliott Matchett, Desmond Mackell, Austen Lorenz, Andrea Mott, Mark Herzog, Josh Ackerman. *"Machine learned daily life history classification using low frequency tracking data and automated modelling pipelines: application to North American waterfowl".* Movement Ecology. https://movementecologyjournal.biomedcentral.com/articles/10.1186/s40462-022-00324-7. 16 May 2022.
41. Ian Goodfellow, Yoshua Bengio, Aaron Courville. *"Deep Learning. Chapter 10: Sequence Modeling: Recurrent and Recursive Nets".* MIT Press. https://www.deeplearningbook.org/contents/rnn.html. 2016.

42. P. Palangpour, G. K. Venayagamoorthy, K. Duffy. *"Recurrent Neural Network Based Predictions of Elephant Migration in a South African Game Reserve"*. IEEE. https://ieeexplore.ieee.org/document/1716662. October 2006.
43. Dhanushi A. Wijeyakulasuriya, Elizabeth W. Eisenhauer, Benjamin A. Shaby, Ephraim M. Hanks. *"Machine learning for modeling animal movement".* https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0235750. July 2020.
44. Kehinde Owoeye. *"Forecasting Avian Migration Patterns using a Deep Bidirectional RNN Augmented with an Auxiliary Task".* Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. https://discovery.ucl.ac.uk/id/eprint/10120848/. July 2020.
45. Takuya Maekawa, Kazuya Ohara, Yizhe Zhang, Matasaburo Fukutomi, Sakiko Matsumoto, Kentarou Matsumura, Hisashi Shidara, Shuhei J. Yamazaki, Ryusuke Fujisawa, Kaoru Ide, Naohisa Nagaya, Koji Yamazaki, Shinsuke Koike, Takahisa Miyatake, Koutarou D. Kimura, Hiroto Ogawa, Susumu Takahashi, Ken Yoda. *"Deep learning-assisted comparative analysis of animal trajectories with DeepHL".* https://www.nature.com/articles/s41467-020-19105-0. 20 October 2020.
46. Uber Technologies. *"kepler.gl"*. https://kepler.gl/. Accessed July 2023.
47. Jeff A. Tracey, Jun Zhu, Kevin R. Crooks. *"Modeling and inference of animal movement using artificial neural networks".* https://link.springer.com/article/10.1007/s10651-010-0138-8. 6 April 2020.
48. Brett T. McClintock, Devin S. Johnson, Mevin B. Hooten, Jay M. Ver Hoef, Juan M. Morales. *"When to be discrete: the importance of time formulation in understanding animal movement"*. https://link.springer.com/article/10.1186/s40462-014-0021-6. 15 October 2014.
49. Movable Type. *"Calculate distance, bearing and more between Latitude/Longitude points."* https://www.movable-type.co.uk/scripts/latlong.html. Accessed 22 July 2023.
50. Seaborn: Statistical Data Visualization; Official Documentation. *"seaborn.kdeplot"*. https://seaborn.pydata.org/generated/seaborn.kdeplot.html. Accessed July 2023.
51. Sunil Kappal. *"Data Normalization using Median & Median Absolute Deviation (MMAD) based Z-Score for Robust Predictions vs. Min – Max Normalization".* https://www.academia.edu/download/61796863/Data_Normalization_Using_MAD_Z-Score20200115-15558-1ttyo6e.pdf. 15 January 2020.
52. Sepp Hochreiter. *"The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions".* International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. https://doi.org/10.1142/S0218488598000094. 1998.
53. Philipp Probst, Anne-Laure Boulesteix, Bernd Bischl. *"Tunability: Importance of Hyperparameters of Machine Learning Algorithms".* Journal of Machine Learning Research. https://www.jmlr.org/papers/volume20/18-444/18-444.pdf. March 2019.
54. Sepp Hochreiter, Jürgen Schmidhuber. *"Long Short-Term Memory".* https://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Term-Memory. 15 November 1997.
55. Ian Goodfellow, Yoshua Bengio, Aaron Courville. *"Deep Learning. Chapter 6: Deep Feedforward Networks".* MIT Press. https://www.deeplearningbook.org/contents/mlp.html. 2016.

56. Official PyTorch Documentation. *"torch.nn.RNN".* https://pytorch.org/docs/stable/generated/torch.nn.RNN.html. Accessed 24 July 2023.
57. Roger Grosse. *"Lecture 15: Exploding and vanishing gradients".* University of Toronto Computer Science. https://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf. 2017
58. Mohsen Fayyaz, Mohammad Hajizadeh Saffar, Mohammad Sabokrou, Mahmood Fathy, Reinhard Klette, Fay Huang. *"STFCN: Spatio-Temporal FCN for Semantic Video Segmentation".* https://www.researchgate.net/publication/306377072_STFCN_Spatio-Temporal_FCN_for_Semantic_Video_Segmentation. August 2016.
59. Phong Le, Willem Zuidema. *"Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs".* https://arxiv.org/abs/1603.00423. 1 March 2016.
60. Seol-Hyun Noh. *"Analysis of Gradient Vanishing of RNNs and Performance Comparison".* https://doi.org/10.3390/info12110442. 25 October 2021.
61. Ian Goodfellow, Yoshua Bengio, Aaron Courville. *"Deep Learning. Chapter 5: Machine Learning Basics".* MIT Press. https://www.deeplearningbook.org/contents/ml.html. 2016.
62. Léon Bottou. *"Stochastic Gradient Learning in Neural Networks".* https://leon.bottou.org/publications/pdf/nimes-1991.pdf. 1991.
63. Christian Darken, John Moody. *"Note on Learning Rate Schedules for Stochastic Optimization".* https://proceedings.neurips.cc/paper_files/paper/1990/file/18d8042386b79e2c279fd162df0205c8-Paper.pdf. 1990.
64. Diederik P. Kingma, Jimmy Ba. *"Adam: A Method for Stochastic Optimization".* Proceedings of the 3rd International Conference on Learning Representations (ICLR). May 2015.
65. Yanzhao Wu, Ling Liu. *"Selecting and Composing Learning Rate Policies for Deep Neural Networks".* https://dl.acm.org/doi/full/10.1145/3570508. 16 February 2023.
66. Michael Rotman and Lior Wolf. *"Shuffling Recurrent Neural Networks".* https://doi.org/10.1609/aaai.v35i11.17136. 18 May 2021.
67. Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner. *"Gradient-Based Learning Applied to Document Recognition"*. https://doi.org/10.1109/5.726791. 1998.
68. PyTorch; Official Documentation. *"torch.nn.CrossEntropyLoss".* https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Accessed 24 July 2023.
69. Scikit-Learn; Official Documentation. *"sklearn.metrics.accuracy_score".* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html. Accessed 25 July 2023.
70. Scikit-Learn; Official Documentation. *"sklearn.metrics.precision_recall_fscore_support".* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html. Accessed 25 July 2023.

71. Scikit-Learn; Official Documentation. *"sklearn.metrics.confusion_matrix".* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. Accessed 25 July 2023.
72. Uber Technologies. kepler.gl; Official Documentation. *"Cluster".* https://docs.kepler.gl/docs/user-guides/c-types-of-layers/f-cluster. Accessed August 2023.
73. Imrus Salehin, Dae-Ki Kang. *"A Review on Dropout Regularization Approaches for Deep Neural Networks within the Scholarly Domain".* https://www.mdpi.com/2079-9292/12/14/3106. 17 July 2023.
74. European Union Copernicus Program and ECMWF. *Climate Data Store (CDS) API: Available Datasets.* https://cds.climate.copernicus.eu/cdsapp#!/search?type=dataset. Accessed 19 July 2023.
75. Ángel M. Felicísimo, Jesús Muñoz , Jacob González-Solis. *"Ocean Surface Winds Drive Dynamics of Transoceanic Aerial Movements".* https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0002928. 13 August 2008.
76. Andrés Viña, Wei Liu, Shiqiang Zhou, Jinyan Huang, Jianguo Liu. *"Land surface phenology as an indicator of biodiversity patterns".* https://www.sciencedirect.com/science/article/abs/pii/S1470160X1600011X. May 2016.
77. Nathalie Pettorelli, Sadie Ryan, Thomas Mueller, Nils Bunnefeld, Bogumila Jędrzejewska, Mauricio Lima, Kyrre Kausrud. *"The Normalized Difference Vegetation Index (NDVI): unforeseen successes in animal ecology".* https://doi.org/10.3354/cr00936. 2016.
78. Ibrahim Sanad, A. Y. El Raffie, F. Altohamy, Mohamed Zayan, *"Tradeoffs for Selecting Orbital Parameters of an Earth Observation Satellite".* https://www.researchgate.net/publication/311486353_Tradeoffs_for_Selecting_Orbital_Parameters_of_an_Earth_Observation_Satellite. 31 May 2012.
79. R. P. Larkin, J. R. Torre-Bueno, D. R. Griffin, C. Walcott. *"Reactions of migrating birds to lights and aircraft".* https://doi.org/10.1073/pnas.72.6.1994. 1 June 1975.
80. Travis Longcore, Catherine Rich. *"Ecological light pollution".* https://doi.org/10.1890/1540-9295(2004)002[0191:ELP]2.0.CO;2. 1 May 2004.
81. NOAA / Earth Observation Group (EOG). *"VIIRS Daily Mosaic".* https://ngdc.noaa.gov/eog/viirs/download_ut_mos.html. Accessed 2 August 2023.
82. NASA Goddard Space Flight Center. *"NASA's Black Marble".* https://blackmarble.gsfc.nasa.gov/. Accessed 3 August 2023.
83. Rien Reijnen, Ruud Foppen. *"The Effects of Car Traffic on Breeding Bird Populations in Woodland. IV. Influence of Population Size on the Reduction of Density Close to a Highway".* https://www.jstor.org/stable/2404646. August 1995.
84. Christopher J. W. McClure, Heidi E. Ware, Jay Carlisle, Gregory Kaltenecker, Jesse R. Barber. *"An experimental investigation into the effects of traffic noise on distributions of birds: avoiding the phantom road".* https://royalsocietypublishing.org/doi/full/10.1098/rspb.2013.2290. 22 December 2013.
85. NASA Earthdata. *"Global Roads Open Access Data Set, Version 1 (gROADSv1)".* https://cmr.earthdata.nasa.gov/search/concepts/C1000000202-SEDAC.html. Accessed 2 August 2023.

86. Douglas M. Hawkins. *“The Problem of Overfitting”.* https://pubs.acs.org/doi/full/10.1021/ci0342472. 2 December 2003.
87. George H. John, Ron Kohavi, Karl Pfleger. *“Irrelevant Features and the Subset Selection Problem”.* https://www.sciencedirect.com/science/article/abs/pii/B9781558603356500234. 13 July 1994.
88. PyTorch; Official Documentation. *“Tutorial: Datasets & DataLoaders; __get_item__”.* https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#getitem. Accessed 24 July 2023.
89. University of Luxembourg. *“HPC @ Uni.lu: The Iris Cluster”.* https://hpc.uni.lu/old/systems/iris/. Accessed 17 July 2023.

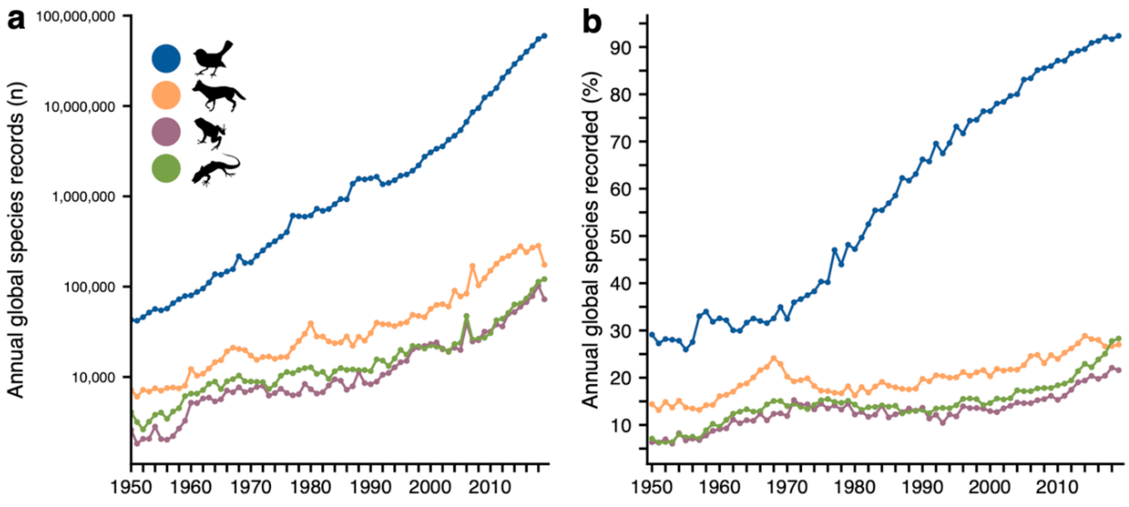
APPENDICES

Appendix 1: Github Links to Project Code

- **[Primary Project]** Machine learning model for classification of animal migrations:
  - **Github:** https://github.com/eio/animal-path-segmentation

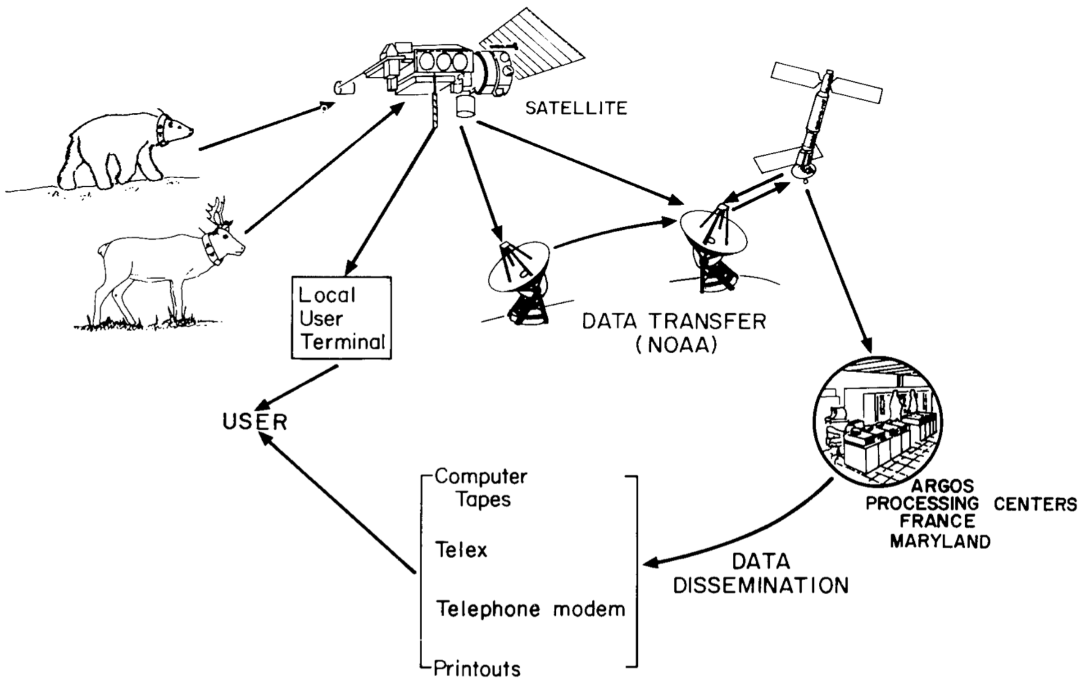
- **[Supplement]** Web-based 3D visualizer for global animal movement data:
  - **Github:** https://github.com/eio/animal-movement
  - **Video demo:** https://www.youtube.com/watch?v=tWhQ8L0fWAk
  - **Live web demo:** (requires Mapbox token) https://eio.github.io/animal-movement

Appendix 2: Increasing Volume of Animal Movement Data

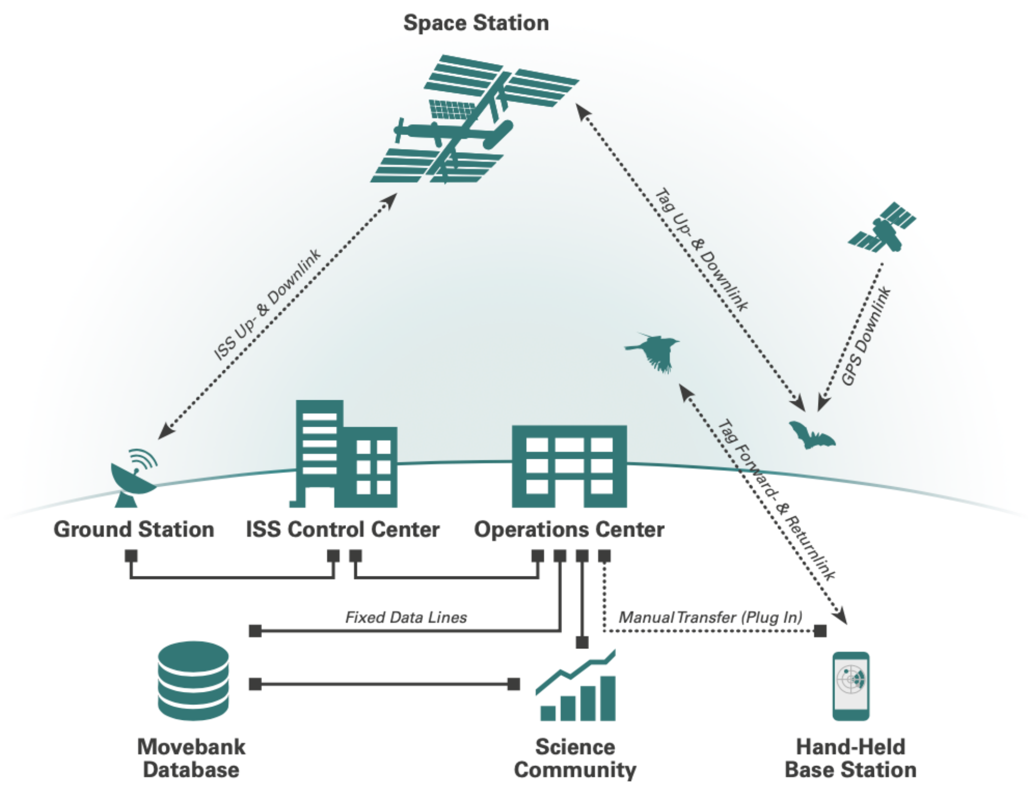


Appendix 2 Fig. A: Annual global species records from 1950 - 2019 [3]

Appendix 3: Space-Based Animal Tracking Architectures



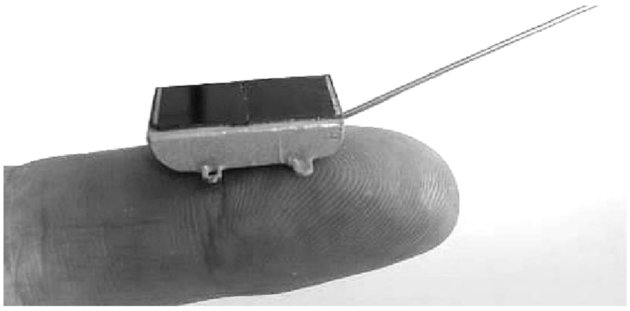
Appendix 3 Fig. A: Overview of the Argos Data Collection and Location System (DCLS) [6].
Included for historical context of an older satellite tracking system.



Appendix 3 Fig. B: Modern ICARUS system architecture, space and ground segments [12].
The ISS segment was replaced with a CubeSat in July of 2023 [16].

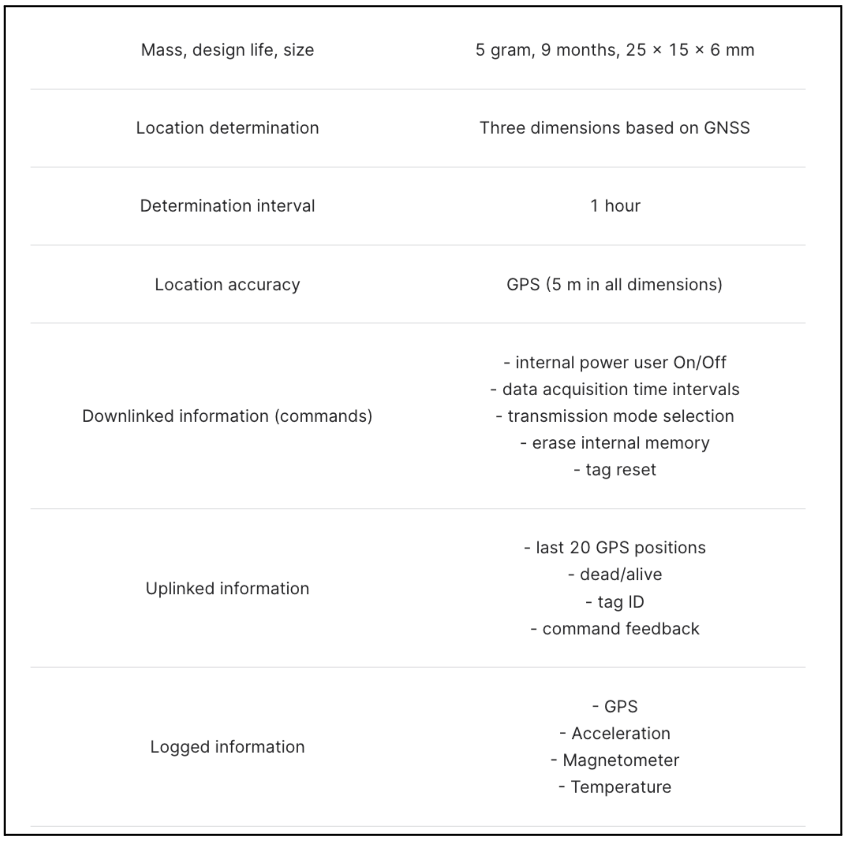
Appendix 4: ICARUS Tag and Communication Details

Appendix 4a: Early ICARUS Tag Design



Appendix 4 Fig. A: Representative sized model of the future < 5 gram ICARUS tag [12].
Devices were built with a 9-month lifespan and dimensions of: 25 x 15 x 6 millimeters
Note: ethical tracking devices should auto-detach when the mission is done.
Ideally, they should also be collected afterwards to minimize pollution.

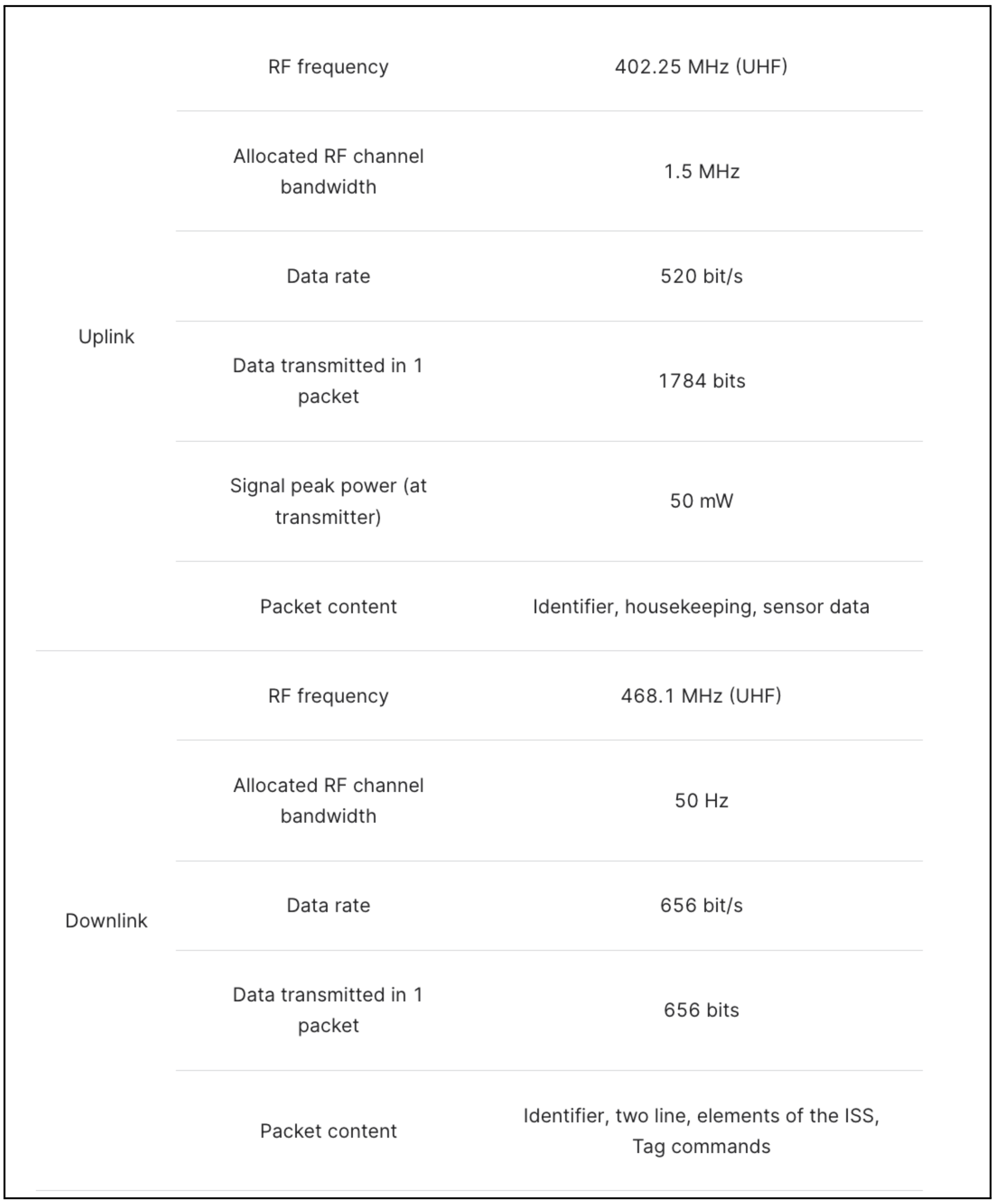
Appendix 4b: ICARUS Tag Key Requirements



Mass, design life, size	5 gram, 9 months, 25 × 15 × 6 mm
Location determination	Three dimensions based on GNSS
Determination interval	1 hour
Location accuracy	GPS (5 m in all dimensions)
Downlinked information (commands)	- internal power user On/Off - data acquisition time intervals - transmission mode selection - erase internal memory - tag reset
Uplinked information	- last 20 GPS positions - dead/alive - tag ID - command feedback
Logged information	- GPS - Acceleration - Magnetometer - Temperature

Appendix 4 Fig. B: ICARUS tag key requirements as of June 2023 [12].

Appendix 4c: ICARUS Uplink and Downlink Specification



Uplink	RF frequency	402.25 MHz (UHF)
	Allocated RF channel bandwidth	1.5 MHz
	Data rate	520 bit/s
	Data transmitted in 1 packet	1784 bits
	Signal peak power (at transmitter)	50 mW
	Packet content	Identifier, housekeeping, sensor data
Downlink	RF frequency	468.1 MHz (UHF)
	Allocated RF channel bandwidth	50 Hz
	Data rate	656 bit/s
	Data transmitted in 1 packet	656 bits
	Packet content	Identifier, two line, elements of the ISS, Tag commands

Appendix 4 Fig. C: ICARUS uplink and downlink specification as of June 2023 [12].

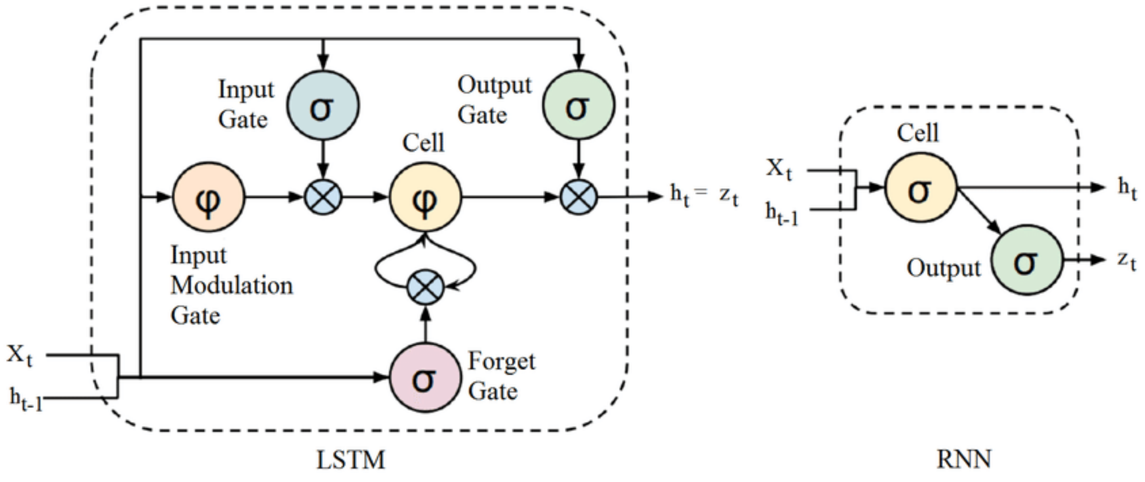
Appendix 5: Code Configuration and Hyperparameter Options

Configuration Option Name	Description
`model`	Base model architecture (RNN or LSTM)
`optimizer`	Base optimizer (SGD or ADAM)
`n_epochs`	Number of epochs to train for
`batch_size`	Input sequence batch size (hardcoded to 1)
`input_size`	Number of model input features (covariates)
`hidden_size`	Number of neurons in each layer (i.e. width)
`num_layers`	Number of stacked RNN or LSTM layers (i.e. depth)
`output_size`	Number of possible output categories (4 seasons)
`dropout`	Dropout rate to randomly deactivate neurons during training, which helps to prevent model overfitting
`weight_decay`	Optimizer weight decay for regularization, another method for preventing model overfitting
`SGD_momentum`	Optimizer momentum for reaching convergence
`init_learning_rate`	Initial learning rate (LR) used by the optimizer
`lr_factor`	Amount of reduction applied by the LR scheduler
`lr_patience`	Number of epochs to wait with no model improvement before reducing via the `lr_factor`
`lr_min`	Minimum LR allowed when using the scheduler
`log_interval`	Number of epochs to wait before printing logs
`plot_every`	Number of epochs to wait before generating loss and accuracy plots during the training process
`save_every`	Number of epochs to wait before saving an output CSV with the model classifications

Appendix 5 Table: Complete set of configuration options and descriptions.
Some options map directly to hyperparameters for PyTorch optimizers and schedulers:

- https://pytorch.org/docs/stable/generated/torch.optim.SGD.html
- https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
- https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

Appendix 6: LSTM and RNN Architectural Differences



Appendix 6 Table A: Examples of a basic LSTM cell (left) and a basic RNN cell (right) [58]

SUPPLEMENTS

Supplement 1: Merging Events and Label Data

After the initial data exploration and cleanup described in Section 2, the next significant processing task was merging the events and label data. To begin, I loaded the large events CSV and multiple smaller label CSVs into two distinct `pandas` DataFrames. This required merging all of the separate label CSVs into a single DataFrame containing labels for every crane.

Then, I explicitly added the implicit states of *"Summer"* and *"Winter"* into the unified labels DataFrame. This was done by inserting new rows into the data structure, after grouping the DataFrame by `Individual` animal. For each record in the labels DataFrame – grouped by individual cranes – I checked the current `Status` string for *"Start"* or *"End"* values. A time delta of one day was added or subtracted from existing `Date` values to determine the new timestamps.

For example, if a row's `Status` value was *"End Spring"*, then I added a new row with a `Status` of *"Start Summer"*. I calculated the new `Date` value by adding a time delta of one day to the old `Date` value. Similarly, if the row's `Status` value was *"Start Spring"*, then I added a new row with a `Status` of *"End Winter"*. In this case, the new `Date` value was calculated by *subtracting* one day from the old `Date` value. I applied the same process for *"End Fall"* and *"Start Fall"* values, resulting in a complete set of eight explicit status strings:

1. *"Start Summer"*
2. *"End Summer"*
3. *"Start Fall"*
4. *"End Fall"*
5. *"Start Winter"*
6. *"End Winter"*
7. *"Start Spring"*
8. *"End Spring"*

The next step was grouping *event* rows by individual crane identifiers, using the `individual_id` column of the events DataFrame. Then, I applied a function across each event record to enact the merge.

For each event `timestamp`, I searched through all of the associated animal's label rows, seeking out the closest `Date` values to that time. I found the encompassing *"Start"* and *"End"* label rows, thereby determining the corresponding event label. I applied this logic to every event of each individual crane, resulting in a brand new event `status` column. This new `status` column omitted the *"Start"* and *"End"* substrings, so each event was labeled with only one of four segmentation classes:

1. *"Summer"*
2. *"Autumn"*
3. *"Winter"*
4. *"Spring"*

These four `status` classes match what is shown in Table 3. They are the basis of the classification performed by the final machine learning model. The output of this step in the process was a single unified CSV that contained all of the geolocation events and labels combined together.

Supplement 2: Model Configuration File and Output Structure

To expedite manual hyperparameter tuning during the training process, I defined every configuration variable in a single file. A few of the settings were hardcoded or pre-determined by other factors, such as the number of input features or class label options. For the settings that were variable and user-defined, an encoded string was generated in the following format, with individual hyperparameters separated by underscores:

```
RNN_SGD_e80_hs128_nl2_d0.2_lr0.001_wd0.001_m0.5
```

The first two items are the base model architecture and optimizer, respectively. The third is the number of training epochs (e). Next are the hidden size (hs), number of layers (nl), and dropout rate (d), all of which are hyperparameters for the ML model. The seventh item is the initial learning rate of the optimizer (lr). The eighth is the optimizer's weight decay parameter (wd). The ninth only applies in the case of an SGD optimizer, and represents the value of the momentum hyperparameter (m).

This encoded string was used as the unique directory name where every generated file was output for each model variation. Within that top-level output directory, subdirectories were created to store: the saved model state after training was finished; the performance evaluation metrics; the final classifications produced by the model in CSV format; and a `config.json` file recording the complete configuration settings.

Supplement 3: Implementing Training and Validation Processes

Loading the Data

The first step in training the model was loading the training and validation datasets. This was achieved with the PyTorch data primitives: torch.utils.data.DataLoader and torch.utils.data.Dataset. The batch_size setting of the DataLoader, while technically a tuneable hyperparameter, was always set to 1 for the purposes of this research. The shuffle setting was set to *True* for the training dataset, and *False* for validation (as well as testing later on). A custom extension of torch.utils.data.Dataset was implemented to handle data loading logic.

During initialization the dataset class loaded, cleaned, and transformed the associated CSV data file; either train.csv, validation.csv, or test.csv, depending on the stage of the workflow. The timestamp column was converted from a string to a Python datetime object, and the DataFrame was ordered chronologically per unique trajectory.

Unique trajectories were defined by concatenating individual_id and year values. Since the data was downsampled previously to single daily geolocations, this ensured that no input sequence was longer than 365 waypoints.

After loading and grouping the data, sequences were converted to Tensors: one distinct Tensor of feature values, and one distinct Tensor of label values.

The input features chosen are described in depth in previous sections, and the final 14 field names were: month, day, sin_time, cos_time, lat, lon, dist_from_prev_loc, velocity, bearing, turn_angle, daily_mean_distance, daily_mean_velocity, daily_mean_bearing, and daily_mean_turn_angle.

The label values, originally strings, were one-hot encoded to numerical vectors. This enabled converting the data structures into Tensors. The mapping of the label string values to encoded vectors was:

```
{
    "Winter": [1, 0, 0, 0],
    "Spring": [0, 1, 0, 0],
    "Summer": [0, 0, 1, 0],
    "Autumn": [0, 0, 0, 1],
}
```

Since the entire dataset could be loaded into memory, the Tensorization happened on load. This meant that it was done during DataLoader creation, rather than inside of the train and test loops. Therefore, no additional processing was required inside of __getitem__, the special Dataset method called when enumerated batches are accessed [86]. This was important for training efficiency, minimizing the time spent during each epoch.

In the `__getitem__` method, the index argument was used to retrieve the unique identifier (i.e. the concatenated `individual_id` and `year` values mentioned above). The unique identifier was then used to retrieve and return the sequence's feature and label Tensors during the training and testing loops.

Training the Model

Once the `DataLoaders` were ready, the script began the training loop. The number of iterations was defined by `cfg.N_EPOCHS`, starting with epoch 1 and continuing through the whole range to N.

The first step in the training loop was calling the `train_process` function, which enumerated the `DataLoader` contents for the training set and iterated through the various batches. As mentioned above, the training `DataLoader` was set to shuffle the order of batches and the hard-coded `batch_size=1` ensured each batch was a single sequence (i.e. trajectory).

For each sequence, the `"features"` and `"labels"` Tensors were accessed with the `Dataset.__getitem__` method. If the hardware device was CUDA-compatible, the Tensors were sent to the GPU (see Supplement 4 for more details on CUDA). Then, the data was passed to another function called `train` where the model made guesses, the loss was computed, and the model was updated via the optimizer.

When using the built-in `RNN` or `LSTM` modules from `PyTorch`, the hidden state is automatically initialized. Therefore, it was not required to call `initHidden()` explicitly. However, the script did call `optimizer.zero_grad()` which cleared the gradients of optimized Tensors to prepare for a new backpropagation pass.

Since the length of the sequence input was variable (i.e. the number of waypoints per trajectory), the sequence length was determined before initiating the forward pass. The sequence length is a necessary input into the `nn.utils.rnn.pack_padded_sequence` utility described above.

The `output_tensor` returned by `model(inputs_tensor, seq_length)` was then used to compute the loss with `nn.CrossEntropyLoss`. The backward pass and optimize steps were performed with `loss.backward()` and `optimizer.step()` and finally the `train` function returned the prediction and loss.

Next, the `output_tensor` was converted from probability values into a list of the category strings: *"Summer", "Autumn", "Winter",* or *"Spring"*. The same was done for the ground-truth labels Tensor, resulting in two ordered lists of such values. The lists of category strings were then compared to determine which model guesses matched the ground-truth labels. The total number of correct predictions per waypoint was counted towards a running tally that stored the overall score across every sequence iteration.

When the loop terminated and all batches from the training set were accounted for, the epoch's accuracy was calculated by dividing the number of correct predictions from `train_loader.dataset.total_records()`. This yielded a decimal value that was multiplied by 100 to obtain the epoch's final accuracy percentage.

Before exiting the `train_process` function and returning the epoch's accuracy and loss, the script saved the current trained state to a local directory. This was done for the model as well as the optimizer and epoch number. In case the top-level training loop was disrupted for any reason, saving the state after every epoch also served as a training checkpoint.

Validating the Model

Back in the top-level loop through every epoch, the next function call was to `test_process`. Much of the logic matched the `train_process` function but some notable differences are worth pointing out. To start, the test loop was inside of `with no_grad():` which informed `PyTorch` that gradients did not need to be calculated for the outputs.

The test loop iterated through the validation set's enumerated `DataLoader` contents *without* shuffling the order of the sequences, and passed each batch to a `test` function which also did a forward pass through the model to make a prediction and compute the loss. The loss value per batch was again stored in an array that was averaged for each epoch. The `output_tensor` produced by the model was converted from probability values to strings, which were compared against the ground-truth string labels. The correct predictions per batch were accumulated to produce the final accuracy metric.

One other important difference within the `test_process` function was the functionality to generate an output CSV of model predictions alongside the ground-truth labels and input features. This was only done for select epochs, determined by the configuration value stored in `cfg.SAVE_PREDICTIONS_EVERY`. To ensure the features were human-readable and not normalized in the CSV output, they first passed through an inverse normalization function. Waypoint rows were stored for each batch (i.e. trajectory) and later combined to create a unified CSV output with all trajectories from the validation set.

Lastly, since the `test_process` function was also used to determine the accuracy of a pre-trained model on the final test dataset, it contained a conditional check to differentiate between a validation run and a test run. If `test_process` was running on the final test dataset, it would also output a predictions CSV and generate additional model performance outputs described in the following section.

Plotting Accuracy and Loss Across Epochs

Following the completion of the training and testing loop iterations for every epoch, where the accuracies and mean losses for each epoch were calculated, four arrays of these performance metrics were ready to be visualized and output:

- `train_accuracies`: List of final accuracies for each training epoch
- `test_accuracies`: List of final accuracies for each validation epoch
- `avg_train_losses`: List of mean loss values for each training epoch
- `avg_test_losses`: List of mean loss values for each validation epoch

These four arrays were visualized in two plots, one for train and test losses and one for train and test accuracies. Both plots were generated using `matplotlib` and both had the number of epochs as the X-axis. The loss plot's Y-axis was the mean cross-entropy loss, and the accuracy plot's Y-axis was the final accuracy percentage. Examples of these loss and accuracy plots are included in the Section 4.

Supplement 4: Final Training and Testing on the HPC System

In order to expedite the model training time and experiments with various configurations, I ran my code on the University of Luxembourg's HPC system (the "Iris Cluster") [87]. I did this to benefit from GPU acceleration via Nvidia Corporation's Compute Unified Device Architecture (CUDA), a parallel computing platform. Using CUDA significantly reduced the time to train the ML model, resulting in a workflow that enabled more efficient hyperparameter tuning.

The first requirement was setting up SSH from a personal laptop to the HPC with public key authentication. I added the Iris Cluster configuration details to my local computer's *`/.ssh/config`* file and uploaded my public key to the HPC website. Then I confirmed an active SSH tunnel between the laptop and the HPC, allowing a command-line session on the laptop to access the HPC filesystem.

Running code on the HPC involved using the Slurm Workload Manager to schedule tasks and allocate resources for specific jobs. This was also required when performing initial installations, in order to access the GPU hardware to properly install `PyTorch` with CUDA. After I checked the latest CUDA version compatible with the GPU hardware, I created a new Conda package management environment and installed all the project dependencies.

I then implemented another simple Slurm script to verify GPU access in the code, using the function *`torch.cuda.is_available()`*. The bash files written for Slurm specified certain configuration options, such as: the name of the job, the output file, the maximum time-limit of the process, the partition or "queue" to execute the job (e.g. one with GPU support), the number of GPU resources required for the job, the number of nodes requested from the HPC cluster, and whether any emails should be sent for different job events (e.g. job start, job end, or job failure). Actively running jobs were checked using *`squeue -u $USERNAME`*, which indicated whether the job was pending ("PD") or if it actually started.

The final components of the Slurm bash scripts were the actual command-line arguments to execute. In my case, it was a three-step process of 1) changing directories to the project code, 2) activating the Conda environment where relevant dependencies were previously installed, and 3) running the Python script to initiate the model training. I ran the Python script with the *"`-u`"* option to enable unbuffered output – without which there would be no way to check print statements until the Slurm job was finished. By specifying the unbuffered option, I could monitor the script's progress in real-time with: *`tail -f $SLURM_OUTPUT_FILE.`*

For each new model training session, I ran two Slurm jobs in sequence. First, I called `python -u run_model.py` to train the model from scratch using the `train.csv` and `validation.csv` files described above. Once that process was fully complete and a new trained model was saved to file, I called the script again with an argument to load the trained model and test on `test.csv`. I implemented this functionality with Python's `argparse` module, adding the "`-l`" option to specify that a trained model should be loaded: `python -u run_model.py -l`.

After the model training and testing processes both completed, I inspected all of the relevant performance evaluation files. I also analyzed the model-produced segmentation labels. To do this, I used Secure Copy Protocol (SCP) to transfer files from the HPC filesystem back to my local computer. There, I reviewed the output visualizations and model classifications.