Cos4Cloud

Co-designed Citizen Observatories Services for the EOS-Cloud

H2020 programme: Research and Innovation action

Deliverable D4.4 - Platform for the interactive pre-processing of camera trap images

27.01.2022, Version 1.2

EUROPEAN COMMISSION

Project funded by the European Commission within the Horizon 2020 Programme (2014-2020)

Grant Agreement No.	863463

Type		
R	Document, report excluding the periodic and final reports	
DEM	Demonstrator, pilot, prototype, plan designs	X
DEC	Websites, patents filing, press & media actions, videos, photos, etc.	
SOF	Software, technical diagrams, etc.	
OTHER	Flyers, etc.	

Dissemination level		
PU	Public, fully open.	X
CO	Confidential, restricted under conditions set out in Model Grant Agreement	
CI	Classified	

Revision history

R#	Date	Description/Reason of change	Deliverable contributors
R0.1	2021.11.12	Create template	Stefan Rueger
R1.0	2021.12.23	First draft	Kai Waddington
R1.1	2022.01.14	Review	Claudia Fabó Cartas and Andreas Matheus
R1.2	2022.01.29	Amendments	Stefan Rueger, Frederic Fol Leymarie and Kai Waddington

Authors

Kai Waddington, Stefan Rueger, Frederic Fol Leymarie (DynAIkon)	Reviewers: Claudia Fabó Cartas (ECSA) Andreas Matheus (Secure Dimensions)

Citation
Cos4Cloud consortium (2022). Kai Waddington, Stefan Rueger and Frederic Fol Leymarie. Platform for the interactive pre-processing of camera trap images (D4.4)

License and attribution
Copyright. All rights reserved.

Executive Summary

This deliverable is a software platform at TRL 8 that processes observations from camera traps. It largely automates a major recurring action for Camera Trap Users to work through thousands of video clips or images that may be empty, filter those observations of interest, propose species names, and - if wanted - upload them to a chosen biodiversity Citizen Observatory, e.g. iSpot, with the proposed identification.

The current best practice for automation of this kind is judiciously deploying specifically trained deep neural networks trained on suitable datasets. We chose the Caltech Camera Traps dataset (Beery et al., 2018) based on size, availability of bounding box information and diversity of images, to train and evaluate different neural architectures with a view to select the best state-of-the-art neural models for species identification. After careful experimentation we identified the Cascade R-CNN X-101 64x4d FPN model (Cai and Vasconcelos, 2019) as most suitable for species identification in camera traps.

We developed an API at https://service.fastcat-cloud.org/api for use by machines and automated workflows to authenticate, upload camera trap images together with a set of species of interest, and to download corresponding annotations. The API is documented at https://service.fastcat-cloud.org/api/spec. The API is also used by our dedicated, interactive web service that allows a user to interactively authenticate, upload camera trap images, choose a machine learning model, download species identification lists, filter images of interest and upload a subset of observations to a citizen observatory.

This document gives the necessary background information, user guide and examples on how to use our deliverable D4.4: Platform for the interactive pre-processing of camera trap images.

Table of Contents

Introduction: Processing of Camera Trap Images **5**

1. Machine Learning in Camera Trap Images **5**

1.1 Data sets 5

1.2 Choosing Machine Learning Models 7

1.3 Training Models for Specific Sets of Animals 10

2. API for Processing Camera Trap Images **11**

2.1 Overview 11

2.2 Authentication and Authorisation 12

2.3 Image Storage 13

2.4 Results Storage 14

2.5 Selection of Machine Learning Models 14

2.6 Direct Upload of Camera Trap Images 15

2.7 Processing and Download of Results 15

2.8 Example Usage 16

2.9 Formal API definition 17

3. Web Interface **20**

3.1 Relation to our API 20

3.2 Authentication and Authorisation 20

3.3 Interactive Filtering 21

3.4 Upload to Citizen Observatories 21

3.5 Walk-through with examples 22

Conclusion and perspectives **30**

Glossary **31**

References **33**

Annex 1 - Understanding Model Names **35**

Introduction: Processing of Camera Trap Images

FASTCAT is a Flexible AI SysTem for CAmera Traps. The aim is to provide state of the art AI services for communities of users of camera traps, with a focus on citizen scientists as well as scientists in need of robust, yet affordable solutions for biodiversity monitoring of animal species. We currently offer two main services: Edge and Cloud. FASTCAT-Edge refers to functionalities offered on the camera trap itself, while FASTCAT-Cloud offers functionalities on the cloud where images taken with a camera trap can be uploaded for processing. This document is concerned with FASTCAT-Cloud. The main functionalities offered are to remove empty images, add bounding boxes around detected animals, and provide some likely species identification. The technology used is based on recent developments made in Machine Learning applied to images.

1. Machine Learning in Camera Trap Images

Current best practice develops Convolutional Neural Network (CNN) models for deriving a semantic meaning from pixels of an image (Petso et al., 2022). This section surveys the state of the art in object recognition, i.e. creating bounding boxes around objects in images and classifying the object within the bounding box. The objects of interest in camera trap images are animals, which restricts the classes in the classification task to animals at the granularity of species, genus or family. Ideally, we want an automated system that takes an image as input and automatically recognises all animals in the image and outputs, for each animal, its corresponding bounding box, an identification suggestion (species, genus or family) and a confidence of this prediction.

1.1 Data sets

Training CNNs, a type of feed-forward neural network, requires datasets typically containing tens of thousands or hundreds of thousands of training examples. The adjustments made to the model parameters during each training iteration are very small and so a correspondingly large number of updates are required. It is ideal to base these updates on a large number of varied training examples to produce a more generalisable model.

When training object detection models, the training data must contain specific annotations: coordinates, or other representation, of the location of each object within the image; and corresponding classifications for each of those objects. The addition of bounding box (bbox) annotations provides a significant increase of the labour and effort cost when annotating an object detection dataset compared to an image-classification dataset, ergo the datasets that are publicly available with this style of annotation are usually more difficult to find, especially for niche themes, as well as containing fewer examples.

We are interested in detecting and identifying animals in images from camera traps and so we need datasets that contain this style of image. We detail several potential datasets in Table 1. We initially exclude the iWildCam datasets from consideration as they are composites of other datasets in the table. We decided to use the Caltech Camera Traps (CCT) dataset (Beery et al., 2018) to create our original models as it has a relatively large number of training examples, and it has a high ratio between the number of examples relative to the number of classes. We decided to also use the Wildlife Conservation Society Camera Traps (WCS) dataset during the early stages of our experimentation as it contains examples from a large number of classes, however, we decided to use the CCT dataset going forward as it contains a greater number of training examples.

Name	*Classes*	*Examples without bbox*	*Examples with bbox*
iWildCam 2020	1.7k	1.65M	450k
iWildCam 2021	204	265k	265k
NACTI	28	3.7M	9k
Caltech Camera Traps	22	245k	66k
Wildlife Conservation Society	675	1.4M	45k
Snapshot Serengeti	61	7.1M	78k

Table 1: Details about various camera trap image datasets.

The CCT dataset contains roughly 66,000 images with the bounding-box level annotations, with these images containing examples from 22 classes. These images were collected from camera traps located in the Caltech region of California. These classes are animal species that are typical to the Caltech region and include animals such as opossums, racoons, badgers, deers, and foxes. In addition, the dataset includes classes for vehicles and empty images which are useful for filtering unwanted images. This dataset contains images captured both during the day, in full colour mode, as well as during the night using infrared mode. Another reason for choosing this dataset is because it is popular in recent literature to make the comparison of results easier.

1.2 Choosing Machine Learning Models

CNNs refer collectively to a category of deep learning models that use the convolution operation. However, there is a large number of different architectures within this category. The architecture of a CNN refers to how the different layers and operations within the model are constructed and connected. Slight tweaks to the architecture can have a large impact on how well the model performs with regards to training as well as speed and generalisability of the resulting model. Deep learning has increased in popularity over the past decade which has driven an increase in the number of new architectures being published. As such, there are a large number of architectures to choose from that have all been shown to provide good results on their respective datasets. Choosing the model architecture will have a significant effect on final results and so the process of choosing the architecture must be carefully considered.

When evaluating object detection networks the most popular metric to use is Mean Average Precision (mAP) (Everingham et al., 2010) and typically the MS COCO (Lin et al., 2014) variant. This metric is a single numerical value in the interval [0, 1] that attempts to quantify the quality of both the predicted bounding boxes (with respect to a ground truth bounding box) as well as their corresponding classifications. The original metric uses a single Intersection over Union (IoU) threshold to determine whether a bounding box is considered correct, i.e., the predicted bounding box overlaps well enough with the ground truth. As the MS COCO variant uses a range of thresholds and averages over each of the resulting mAP scores, this version could be more accurately named the Mean of Mean Average Precisions. In our use-case it is not critical that the bounding boxes are accurate down to pixel-perfect granularity. In our experience it is the classification of the bounding boxes that offers the most room for improvement, with bounding boxes starting to look good at a mAP score of approximately 0.30.

We start by choosing four promising architectures for general image classification tasks and train two versions each: one on the MS COCO dataset with generic object classes (including some animal clases) and one on a camera trap dataset, for which we selected the Wildlife Conservation Society dataset (WCS). The four architectures that we chose were: Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al., 2015), RetinaNet (Lin et al., 2017a), and SSD (Liu et al., 2016). Our reason for choosing these architectures was that they were, at the time of the experiment, providing state-of-the-art results for object detection on the MS COCO dataset. We used the same training configuration intra-architecture for both datasets. This experiment was to find a baseline of what kind of performance we could expect on a camera trap dataset, and also to provide some level of insight into how well the performance of models devised for general object detection being trained on the MS COCO dataset transfers to camera trap datasets.

The results in Table 2 show a consistently worse performance when models are trained on the WCS dataset compared to when models are trained on the MS COCO dataset. One cannot compare results from different datasets directly. However, considering several models that use a variety of techniques displayed the same pattern of a worse performance on the WCS dataset, suggesting that species prediction is more difficult on this particular dataset. The experiment outlined above constitutes our preliminary set of experimentation. Based on this we devised a more useful experiment.

Architecture	*MS COCO 2017*			*WCS*		
	mAP	*mAP^{50}*	*mAP^{75}*	*mAP*	*mAP^{50}*	*mAP^{75}*
Fast R-CNN	**0.359**	**0.557**	**0.390**	0.246	0.354	0.265
Faster R-CNN	0.330	0.515	0.358	0.235	0.346	0.259
RetinaNet	0.343	0.516	0.366	**0.301**	**0.431**	**0.353**
SSD	0.189	0.349	0.186	0.211	0.360	0.252

Table 2: Results from evaluating models trained on the MS COCO and WCS datasets.

For this next experiment we identified seven architectures that currently provide state-of-the-art results for object detection tasks. They are represented in Table 3. Our intentions for this experiment are to find out:

- To what extent model performances drop on camera trap datasets compared to the generic MS COCO dataset.
- If there is an intrinsic property of images of animals that causes this performance degradation.
- What the maximum performance is that we can achieve using so-called off the shelf techniques.

To achieve these goals we trained the architectures on the MS COCO dataset and evaluated them on the dataset as a whole, and on the subset of the dataset that contains only animal classes[1]. We also trained and evaluated separate instances of the architectures on the CCT dataset.

Implementing all of these architectures ourselves would take a lot of time, and so would setting up the experiment multiple times for each different platform that open-source implementations might use. Our solution to this is to use the MMDetection library (Chen et al., 2019). MMDetection is a framework built on top of the MMCV library (MMCV, 2018) created by the same group, which in turn is built on top of Python and PyTorch. It is a competitor to Facebook's Detectron (Girshick et al., 2018; Wu et al., 2019) and provides implementations for many modern architectures to which the community actively contributes — making it a good

[1] This subset of the MS COCO dataset fits nicely with our intentions to focus on animal detection, and we believe it to be a good candidate for a high-quality animal dataset that is readily available.

framework to use. Using this framework saves us the time of implementing many of these architectures ourselves and allows us to use a consistent system of pipelines for data processing. We can set up the data ingestion pipelines once and it will work for the training of the various different architectures.

In addition, MMDetection provides pre-trained models which are trained on the MS COCO dataset, alongside the corresponding configuration files to reproduce these models. In our experience these configurations have been useful to correctly reproduce the models that they provide, and so for this experiment we use the pre-trained models provided by MMDetection when evaluating on the MS COCO dataset. When we trained the architectures on the CCT dataset we tried to keep the training configurations as similar as possible to the architectures trained on the MS COCO dataset, but in some instances we had to reduce the learning rate as training was unstable. We compared the mAP scores (Everingham et al., 2010; Lin et al., 2014) that we achieved using the seven chosen architectures on the CCT dataset to the mAP score that each paper reports for the architecture on the MS COCO dataset as well as to the subset of the MS COCO dataset that contains only animal images.

Architecture	*MS COCO mAP*	*MS COCO mAP (animals only)*	*CCT mAP*
GFL X-101 32x4d Dconv2 (Li et al., 2020; Zhu et al., 2019)	**0.481**	**0.671**	0.173
Deformable DETR R-50 (Zhu et al., 2020)	0.468	0.661	0.343
ResNeSt Cascade R-CNN S-101 FPN (Zhang et al., 2020)	0.468	0.644	0.366
DCN Cascade R-CNN R-101 FPN Dconv(c3-c5) (Dai et al., 2017; Zhu et al., 2019)	0.450	0.641	0.385
Cascade R-CNN X-101 64x4d FPN (Cai et al., 2019)	0.447	0.632	**0.394**
RepPoints X-101 FPN DCN (Yang et al., 2019)	0.442	0.627	0.180
GA Faster R-CNN X-101 64x4d FPN (Wang ey al., 2019)	0.439	0.621	0.379

Table 3: Results from evaluating state-of-the-art architectures on the MS COCO dataset, the animals-only subset of the MS COCO dataset, and the CCT dataset.

Contrary to the results obtained on the WCS dataset we notice an improvement from Column 1 to Column 2, i.e., when evaluated on the MS COCO animal subset compared to the dataset as a whole. We suspect that this might be because all the images in the MS COCO dataset are high-quality, high-resolution, full-colour and sharp, with the subject of the image, in this case the animal, centred and mostly fully-contained in the image, while also being in focus. These are properties that are rarely observed in camera trap datasets such as the CCT or WCS datasets.

In Table 3 we see that the Cascade R-CNN X-101 64x4d FPN is the best performing architecture on the CCT dataset when comparing the models using the mAP metric. As we are primarily interested in good performance on camera trap datasets, we decided to use this architecture and model as one of the models available in the FASTCAT-Cloud service.

1.3 Training Models for Specific Sets of Animals

If we provided only models trained on the CCT dataset then those models would be limited to detecting only the 22 species in the CCT dataset. This would exclude a significant number of users from finding the service useful and thus limit the size on the market for such a service. Ideally, we want to be able to detect every possible species set that a user may be interested in, and at the same time do not want to offer a generic one-model-identifies-all-species network for fear of reduced precision that normally comes with a large number of classes.

The sheer number of possible sets of interest is forbidding, and it is clearly infeasible to proactively create bespoke models in advance. Our approach is to generate these models just-in-time. We contemplate as part of our business model to create bespoke models with the users semi-automatically. This involves allowing a user to specify which species they are interested in, upon which we source training examples from the GBIF[2] dataset to create bespoke models for that set of species. Where needed, we plan using the MegaDetector (Beery et al., 2018) or a similar service to generate bounding boxes for the images from GBIF to provide us with the necessary annotations for training an object detection model. We do not train a new model from scratch at this point. Instead, we fine-tune a model that we previously trained on over 1M images collected from GBIF.

Training models requires expensive hardware and takes a long time. If we could only train and evaluate a single model at a time progress would be stifled. To combat this compute constraint we decided to train models using the AWS infrastructure (from Amazon). We use the MMDetection framework to perform model training and have this software setup inside Docker containers making it easy to transfer to a cloud service, such as AWS EC2.[3] Doing our training in AWS allows us to access better hardware as well as to run multiple instances of the same hardware in parallel, greatly decreasing the time taken to train multiple models.

[2] https://www.gbif.org/
[3] https://aws.amazon.com/ec2/

2. API for Processing Camera Trap Images

This deliverable is a web-based service that provides functionality for pre-processing and filtering of camera trap images being able to reduce batches of observations to contain only species that the user is interested in, or to simply alert the user to the different species present. We offer this ability through a RESTful API running on our own servers, with a no-fee and no-user-account required approach, and also use this API as the backend to our web-interface for the service.

2.1 Overview

The overall approach of our API is one of providing key functionality of uploading visual material to the site, have it processed by AI species ID classification programs, offer to store the uploaded images to enable the generation of persistent URLs of the observations, and potentially share their observations with other services. The key design concept is one of modularity so that other sites can utilise our API within their workflows for powerful enhancements.

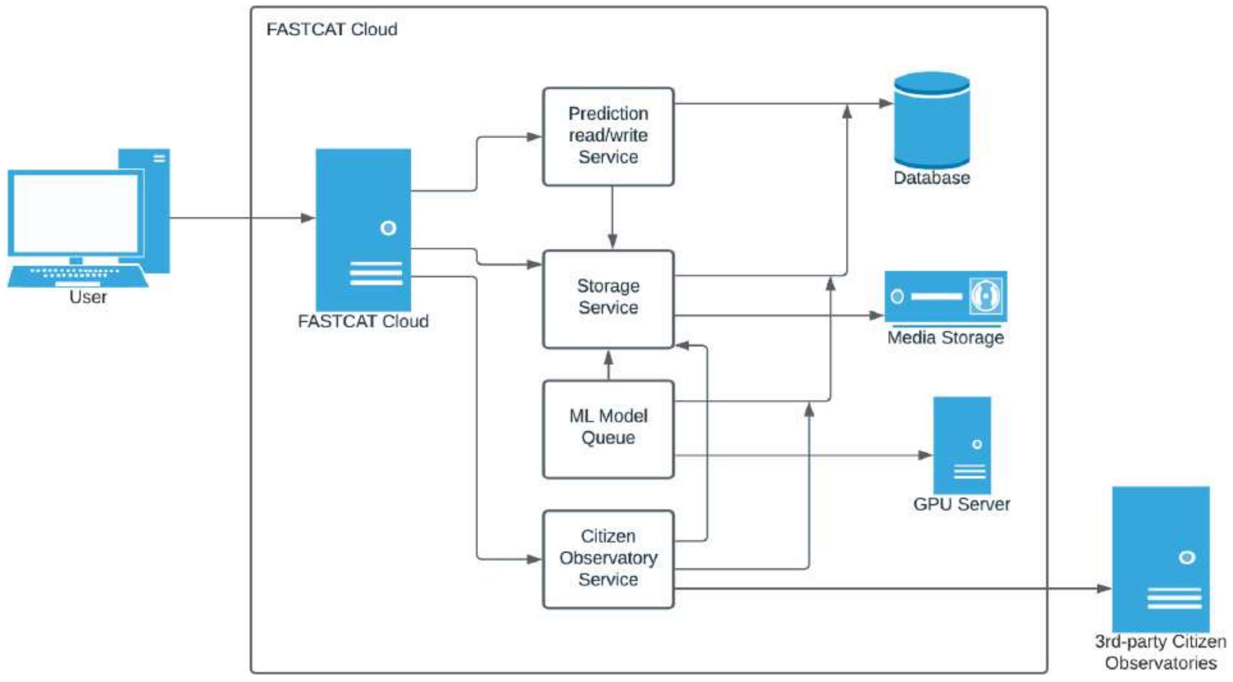


Figure 1: Overview diagram

Figure 1 visualises the different components of our API and how they interact with each other; the prediction service is responsible for all incoming requests pertaining to retrieving results of already processed media as well as handling the setup of the necessary workflows for new media; the storage service handles access to media files on disk, as well as generating public

URLs for media items shared to citizen observatories; the ML Model Queue service manages the compute-intensive task of processing media with any ML models deemed necessary; and finally the Citizen Observatory process is responsible for the integration between our data models and any APIs used to post observations to other citizen observatories.

2.2 Authentication and Authorisation

While we allow users to use the FASTCAT-Cloud API without authentication, when doing so we restrict functionality to fewer observations uploaded in each batch and to more restrictive rate limits when interacting with the API. If a user is authenticated then we relax these restrictions; this functionality allows us to identify and block users abusing our system.

We offer multiple methods of authentication as the API can either be directly used, or it can be accessed via our own web interface. If used via the web interface, we adopt an OAuth2 workflow to authenticate users with either Authenix or Google as the identity providers. The authentication information is stored in a browser session which is included with subsequent API requests. If the API is instead directly used, then we authenticate a user by means of a unique key, generated through the web interface, as a URL query parameter on all requests. The OAuth2 authentication workflow will be discussed further in Section 3.2.

After a user has created their account through the web interface[4] they will have access to a page where they can view, create, and delete API keys associated with their user account. These keys are unique to them and there is no limit to how many they can create or have existing at any time. There is a many-to-one relationship between API keys and user accounts. This allows users to have multiple applications/devices submitting data to their account each using a separate key making their account more secure: if an API key is compromised then the user needs only to delete, regenerate, and replace the details for that single key.

Once a user has obtained an API key[5] they can directly use the API with the same lessened restrictions that they would have access to through the browser. They do this by appending *apiKey=<my key>* and *userId=<user ID>* as a query parameter to any subsequent API requests that they need to be authenticated for, where *<my key>* is their API key. When a request that requires authentication is received by the API, we look for this query parameter and, if present, we attempt to match the API key to an existing user account. If a corresponding user account is found then this user object is added to the request internally and can be used by any subsequent business logic. If no matching user account is found we return an error response to the user after a small delay.

[4] A user registers an account with us through Google or Authenix, from which we use the name and email address to create a browser session tying it to our user record. That session is short-lived and included in subsequent requests from the browser to our API. We store an account for users saving elements such as iSpot User ID and API keys. When a user logs on via Authenix they are shown an authorisation screen from Authenix that clearly states what information will be provided to FASTCAT-Cloud.
[5] More details at https://service.fastcat-cloud.org/api/spec and in sections below

2.3 Image Storage

Users upload media associated with their account and run compute-intensive operations, i.e. deep learning models, against these media items, both requiring consideration to protect the system from unfair-usage-behaviours or abuse. If a user is allowed to upload an unlimited number of media items then they could crash the application by occupying all storage space on the hardware, and if they were allowed to trigger an unlimited amount of predictions against their media then they could make the system essentially unusable by hogging all available compute resources, thus not allowing other users to use the system in the intended way.

Another important consideration is being able to mitigate the potential to store adverse or unwanted media items on our hardware. In this scenario unwanted refers to such items that might be violations of the law or cause distress to those who view them.

When a user uploads media to our API we first detect the filetype of all items and return an error to requests that contain file types that are not on our list of approved types or that are above a certain size. All media items that pass this check are stored in a temporary storage directory. Some actions only require the media to be there for a very short time, for example when the user only intends to run a species ID classification. They might later decide to upload a subset of the provided media to a Citizen Observatory (CO) as a nature observation upon which the temporarily stored media will need to be stored persistently. There is a job that automatically runs periodically that attempts to clean this temporary storage directory by deleting any items that are above a certain age; it is then assumed that they are safe to be deleted.

The next step of the media storage process is to attempt to move media items to a more permanent storage location when needed. Media items arrive at the API through a request and a later part of that same request handles this storage process. The locations of the files at their temporary storage location are handed-off to our storage service. This service retrieves each of the files and creates records in the database for them, which are tied to a user account ID, and moves them to a */YYYY/MM* directory structure for performance benefits. The new locations of the files are also stored in a database with a numeric reference created for each media item.

There is also a job that runs periodically to delete old otherwise unused media items as we do not intend to act as a permanent media storage service for all uploads; we intend to store media only long enough for the results to be beneficial to users.

So far we have covered storage of media items that are accessible only within the application itself, however, we also offer the functionality to registered users wishing to post their observations to iSpot, to create permanent externally-accessible URLs for media items. Users may wish to submit media and classification results to 3rd-party COs. However, many of the COs that we may integrate do not accept media items directly; instead they accept URLs to media items which they will download themselves. If a user wishes to submit their results to a

CO then we will generate a permanent URL for that item and provide it to the user and the CO. To do this we have another service that serves files from a particular location. When we want to generate a public URL to a media file we copy that file to this other location and add a record in the database to signify that we have done so. We can deduce the public URL for this media item based on the location the file was copied to.

2.4 Results Storage

Media items are uploaded with the intention of being processed by our deep learning models which produce information about the media. The output of these models, and the type of data that they produce, varies greatly and does not conform to a fixed schema. We reformat such information into a flexible JSON structure and store the entire JSON string in a single record in a database. That record is related to a media item with a many-to-one relationship which allows for multiple model classifications for each single media item; this is important as it allows us to let users re-process observations with different or newer models without losing the data produced by previous models.

We chose this approach as it is likely that when adding more models in the future, or updating models, the format of the output will vary by too much to be able to account for this now and produce a fixed schema. We instead delegate the responsibility of parsing the data to the consumer of the data.

2.5 Selection of Machine Learning Models

As previously mentioned we can provide access to multiple models through our API. The current state of the FASTCAT-Cloud service allows users to choose from our selection of models at the time of upload. This method of letting the user choose the model to run is based on information about what the model does. In the future we intend to allow different methods of model selection such as choosing species of interest and automatically decide which best models to run to achieve those results.

As we are working on methods to automatically generate new models, we will also make these available through our service as and when they are generated.

2.6 Direct Upload of Camera Trap Images

With camera trapping there is a delay in accessing data and the details about the data: there is the delay of deploying the camera trap and leaving it deployed for an extended period of time before it is retrieved and the observations downloaded from its internal memory, and then there is another delay from when the observations are processed. FASTCAT-Cloud already helps with minimising that second delay, but what if you could access results about the data in almost real-time? We created an integration between our own smart video camera trap, FASTCAT-Edge, and this service, FASTCAT-Cloud.

Our smart video camera trap can automatically upload collected observations directly to an account on FASTCAT-Cloud should the user want to. To do this they would need to register an account, create an API key, and then include this API key and their user ID in the FASTCAT-Edge configuration on the smart camera trap.

If the user sets up this integration then the collected observations will be uploaded from our smart camera trap to our web service, at which point our model pipeline will run on the observations, and the species detections will be almost-immediately available[6] in the user area. This allows users of our smart camera trap to access species information about detections far sooner than other methods allow. In the future we may offer the ability to receive a notification of various types that can be triggered by detection of specific species.

2.7 Processing and Download of Results

Deep learning models are compute intensive and can be relatively slow, taking more than 0.5 s to process each image, even on the high-end consumer-grade hardware that we have. This is problematic as users typically demand a responsive experience with a fast turn around. To address this we created the API with a batch-processing mode.

The batch-processing mode is the default mode and adds all incoming uploads and requests to the end of a queue, returning a unique ID for this request to the user. The user can use this ID in a later request to retrieve the results when they are ready. We have a separate process on the server that monitors the database for new requests and actions them accordingly. This allows us to provide both a responsive experience to the user as well as making us more resilient to system failures enabling us to more easily resume service in the event of such a scenario. Once a record in the request queue has been processed with all of the models required, the results are stored with a relation to the media items and are now able to be retrieved using the ID returned to the user at the time of upload.

We plan to add a real-time mode to the API where requests are added to the top of the queue and results are returned directly to that request rather than an ID which can be used to retrieve results in a subsequent request. We will have to consider how to stop this mode being abused.

2.8 Example Usage

Let us go through an example of a usage story to illustrate the benefits of using the FASCAT-Cloud service. Let's assume that our user is someone that has an interest in the wildlife at some specific location, and to monitor it they decide to deploy a camera trap. After two weeks, they collect the camera trap and copy all of the recordings to their computer. They now have a directory of 30,000 images. If we consider the Snapshot Serengeti dataset as a reference (Swanson et al., 2015), then we can expect that about 70% of those 30,000 images

[6] Delay between the time of the upload and the time until results are available depends on how busy the service is at the time of the upload; observations will be added to the request queue to be processed.

will be empty of any animals, i.e. 21,000 images are of no interest. If one could then somehow iterate through all of the images and sort them manually as empty vs not-empty at a rate of 1 second per image, assuming no breaks, this will take over 8 hours of cleanup. At this point you still need to go through the remaining 9,000 images and add bounding boxes as well as species labels. This boils down to a tedious and prone-to-errors process; adding the species information requires enough expert knowledge of the expected species to be able to distinguish them.

An alternative to processing all of these images manually would be to use the FASTCAT-Cloud API. The user can for example write a simple script to iterate over all of the images and submit them to our API to get results back. The results that we return will indicate if the image is empty, in which case the user can discard it, or it will return the species that it contains, in which case the user can organise the images into different folders. We also provide the bounding box information and so the user can have boxes drawn onto the images as well. We provide an example of NodeJS code to do this below in Code Sample 1.

```
import { readdir, readFile } from 'fs/promises';
// Where are the images stored
const imageDirectory = '/User/Pictures/Camera-Trap-1/images/';
let fastcatCloudResults;

try {
    // Get a list of all of the images
    const imageNames = await readdir(imageDirectory);

    // Send each image individually to FASTCAT-Cloud
    fastcatCloudResults = await Promise.all(imageNames.map((imageName) => {
        // Read the image file
        const image = await readFile(`${imageDirectory}${imageName}`)

        // Construct the data to send to the API
        const formData = new FormData()
        formData.append('image', image)

        // Make the HTTP request
        const response = await
fetch(`https://backend.fastcat-cloud.org/api/v2/predictions/demo`, {
            method: 'POST',
            headers: new Headers(),
            body: formData,
        })

        // Parse the response
        if (response.ok) {
            const jsonResponse = response.json();
            const results = JSON.parse(jsonResponse.body.results)
            return results
        }
        return undefined
    }));
} catch (err) {
    console.log(`There was an error: ${err}`)
}
```

Code Sample 1: Example NodeJS code that could be used to submit images to the FASTCAT-Cloud API. This should be considered as pseudo-code.

The NodeJS sample above should be considered as pseudo-code and used as a guideline rather than an actual script to run. Some improvements would be to batch multiple images into the same request. In the unauthenticated mode users can add 10 images in a single request. In addition, better response and error handling should be added so that the script does not exit because of an error.

The exact specification of request and response formats can be found in our format API specification which is described in more detail next.

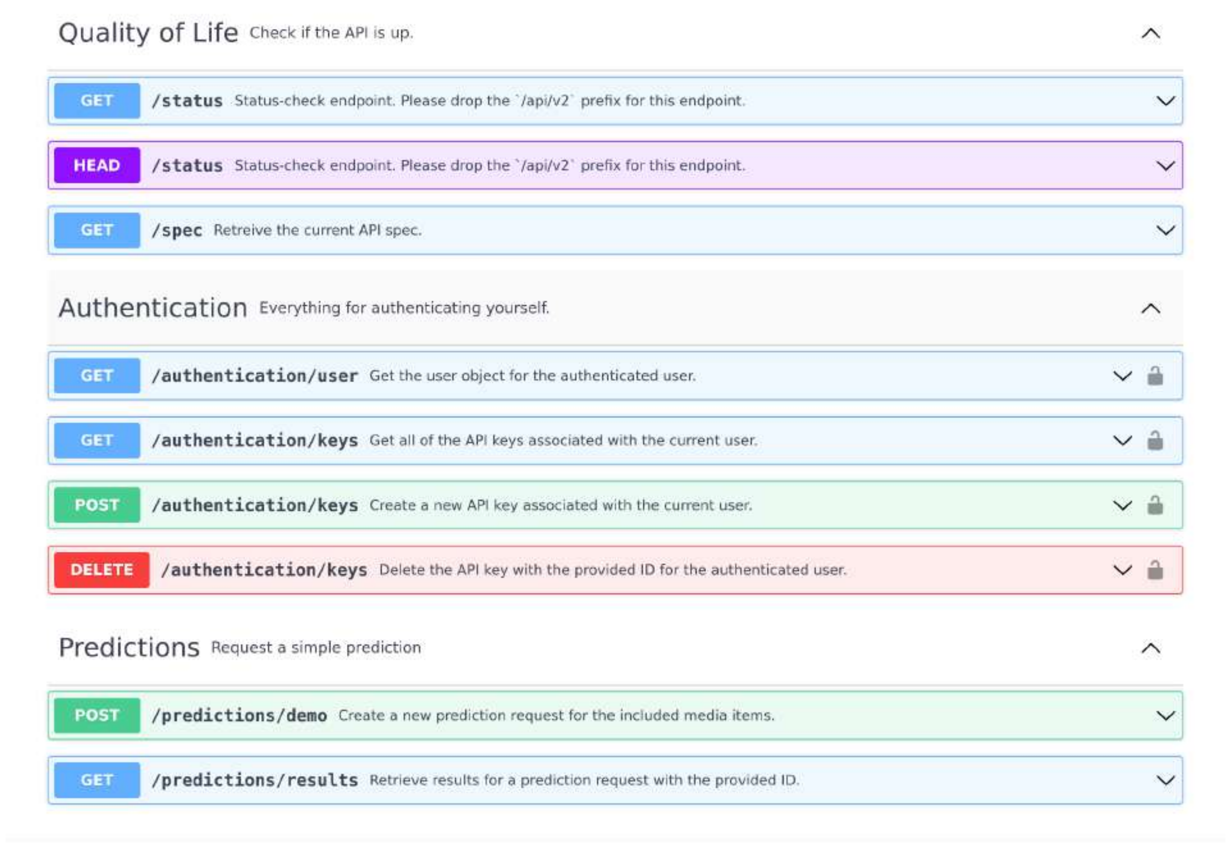
2.9 Formal API definition

The FASTCAT-Cloud API is designed to be mostly RESTful, meaning that the HTTP methods used for each request influence what that request does. There can be different implementations for different request methods to the same endpoint to achieve different results. An example could be a GET and a DELETE request to the same endpoint to either retrieve or delete that user information.

As APIs can be large it is important to document their format, what each endpoint is, what each endpoint should do, the HTTP methods that are accepted, the request formats, and required data for the request, the different response codes, the different response formats, authentication, etc. This is important so that any user can develop their application using our API, and they can be sure that they have handled all of the possible scenarios that our API will present to them.

We use the OpenAPI[7] format for documentation. It defines a consistent format that can be used to describe an API. Moreover, there are tools available that will take this specification document and present it nicely, as well as allowing users to test out the API directly in the browser without having to write any code. An example of the visualisation of this formatted specification can be seen below (Figure 2). Clicking on any of the boxes for an endpoint will give the user more information about that endpoint, such as any properties or parameters that it expects or requires, as well as the various response codes and formats that the user can expect.

[7] https://swagger.io/specification/


Quality of Life Check if the API is up.

GET /status Status-check endpoint. Please drop the `/api/v2` prefix for this endpoint.

HEAD /status Status-check endpoint. Please drop the `/api/v2` prefix for this endpoint.

GET /spec Retreive the current API spec.

Authentication Everything for authenticating yourself.

GET /authentication/user Get the user object for the authenticated user.

GET /authentication/keys Get all of the API keys associated with the current user.

POST /authentication/keys Create a new API key associated with the current user.

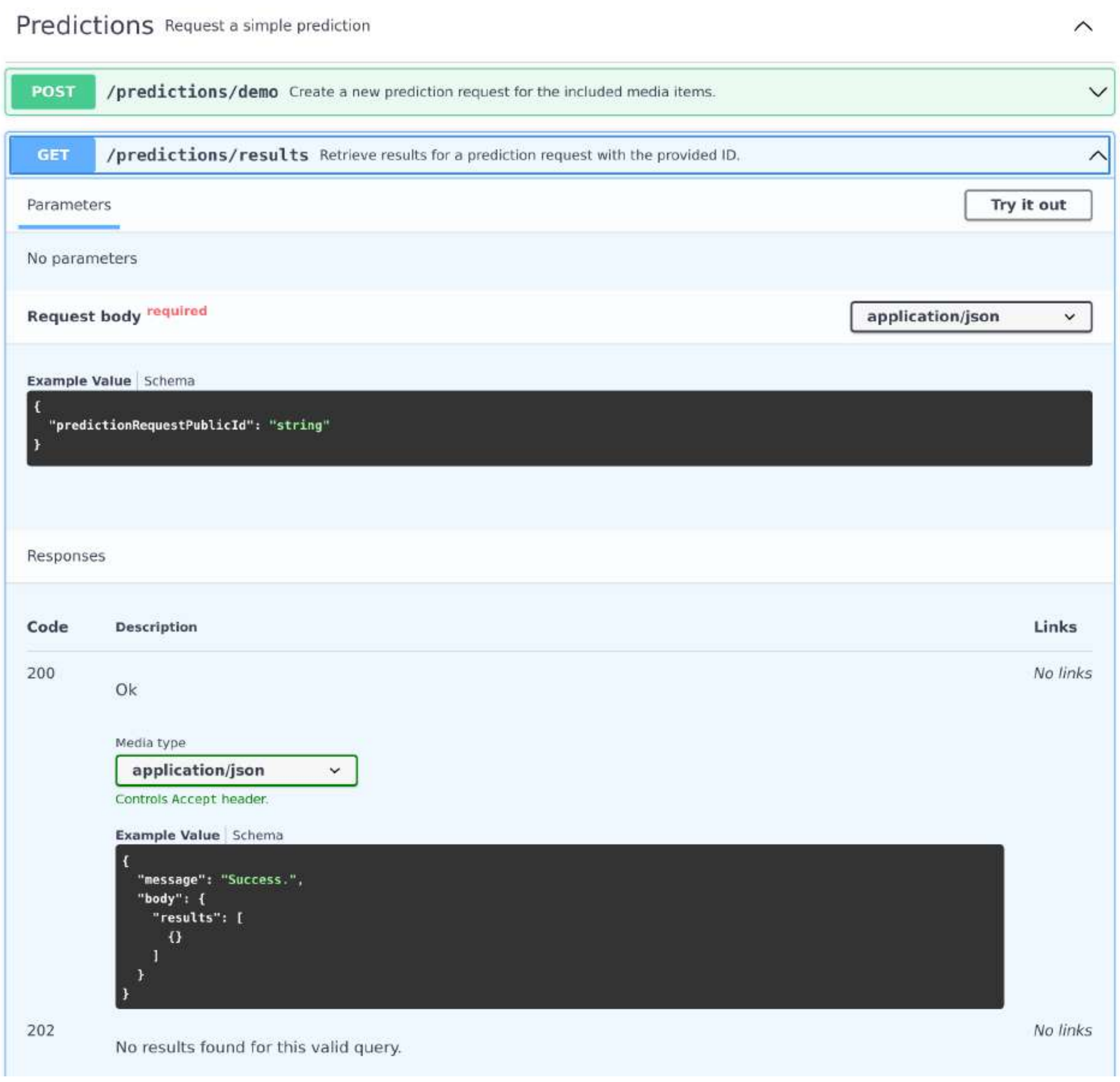
DELETE /authentication/keys Delete the API key with the provided ID for the authenticated user.

Predictions Request a simple prediction

POST /predictions/demo Create a new prediction request for the included media items.

GET /predictions/results Retrieve results for a prediction request with the provided ID.

Figure 2: This figure shows an overview of the visualisation of our public API specification. Each API endpoint with its corresponding HTTP request method can be seen, as well as whether authentication is required for that specific endpoint or not. See https://service.fastcat-cloud.org/api/spec/ for the most up-to-date version of the API specification.


Predictions
Request a simple prediction
POST
/predictions/demo
Create a new prediction request for the included media items.
GET
/predictions/results
Retrieve results for a prediction request with the provided ID.
Parameters
Try it out
No parameters
Request body required
application/json
Example Value
Schema

```
{
  "predictionRequestPublicId": "string"
}
```

```
{
  "message": "Success.",
  "body": {
    "results": [
      {}
    ]
  }
}
```

Figure 3: This figure shows more information about the /predictions/results endpoint.

The latest version of our API specification can be found on the FASTCAT-Cloud website at https://service.fastcat-cloud.org/api/spec, which will be displayed in the format shown in Figures 2 and 3, or the YAML file for the specification can be retrieved directly from the API by sending a GET request to https://backend.fastcat-cloud.org/api/v2/spec (opening this URL in your browser will achieve this) which can be viewed in an online viewer such as the one found at https://editor.swagger.io/.[8]

[8] Our specification may display some errors if viewed in an online viewer such as the one at https://editor.swagger.io/; however, these are not indicative of mistakes in the specification or our implementation of the specification and instead are attributed to limitations of the specification format.

Our entire FASTCAT-Cloud API is actually significantly larger than what is described in the specification; we have excluded endpoints that a user does not need to use, such as those used for authentication only by the FASTCAT-Cloud web interface.

3. Web Interface

3.1 Relation to our API

Using an API directly, programmatically, requires an above average amount of technical knowledge. If we were to limit interaction with our system through offering only the API, we would exclude a large number of potential users. To address this, we created a user-friendly web interface that permits access to the same functionality provided by our API, but without requiring that technical know-how. If a user wants to process only a few images occasionally then it is probably not worth their time to write the code to do so using the API; instead they would likely prefer to use the web interface.

At a technical level the web interface uses the exact same API under-the-hood and so it merely serves as a way to access the API functionality through a graphical interface.

3.2 Authentication and Authorisation

As the API restricts usage heavily for users that are not authenticated, we provide the ability to authenticate a user through the browser. We do this using the OAuth2 protocol and offer this ability using two identity providers: Google and Authenix[9]. A user can navigate to the login page and use either of the identity providers to login, one button each for Google and Authenix authentication modes, and will have to then follow the prompts from the respective identity providers to provide authentication using their existing user accounts with those services.

During this authentication process we receive some information about the user, such as their name and email address, and store this information in our database thus creating a user account for that user. A user account is uniquely identified by its email address and so if a user logs in once using their Google account, and then logs in another time using Authenix with the same email address, we will match those two authentications to the same user account that we have on record.

After a user has been authenticated using this method a unique ID for that user is stored in the browser session storage and this session is attached to any subsequent requests to the API from the web application. The API decodes this session data and attempts to match the user ID to a user record in our database which can be used later by other business logic on our server.

[9] *Authenix* is a project partner part of the Cos4Cloud project. One of their deliverables is a single authentication provider service that can be used commonly amongst all of the other project partners so that users of any of the project deliverables can be granted access to their information in a single location.

3.3 Interactive Filtering

After we have applied our species detection models to media observations we have the opportunity to perform further processing of the images. For example, a user might only be interested in images that contain species A, B, and C.

This can be achieved in a few ways. The first way is for the user to download a JSON file containing the species' information for all observations. This puts the burden of the filtering/sorting of observations on the user, but it is the simplest to achieve. The user can later decide to filter on different species as they would have the useful data on all observations.

Another option is for the user to download a JSON results file containing only the relevant detections. This provides a smaller, and thus more manageable file for the user which they can use to organise their media observations. Similar to above, the burden of sorting would still be on the user, however, in this scenario they can iterate through all of the images referenced in the results file and organise them accordingly.

3.4 Upload to Citizen Observatories

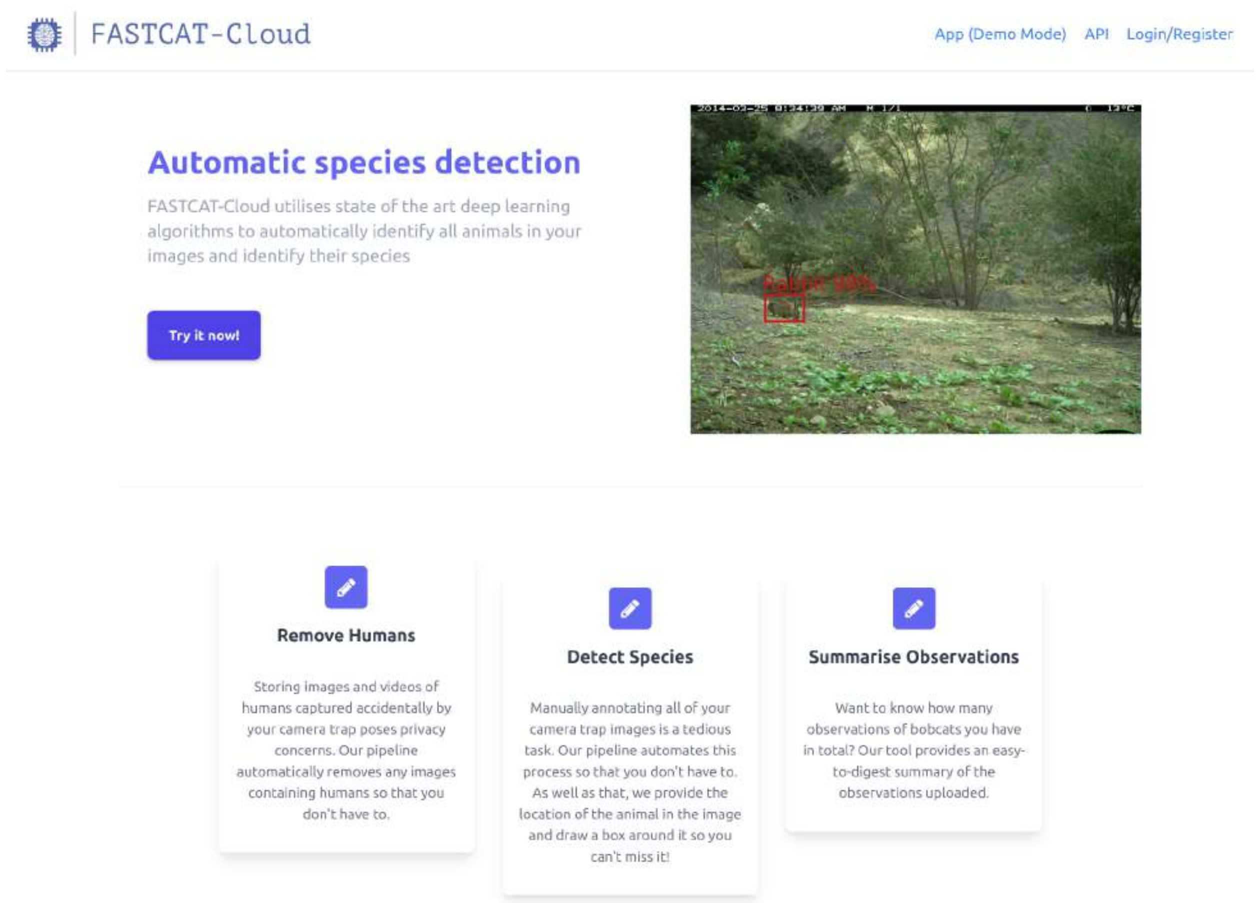
As part of our deliverable we integrate with other Citizen Observatories (COs) and partners of the project. Our first integration with a CO that we offer is with iSpot, https://www.ispotnature.org/. iSpot is a social platform designed around the sharing of wildlife observations. Members of the platform can submit observations, as well as suggest species classification for their own, and other peoples', observations. Users can also open discussion with other users, in the form of comments on observations. If they do not wish to register they are still free to search their database of observations.

When a user has uploaded observations to FASTCAT-Cloud, and receives back the results for them, we allow a user the ability to post these media items as well as the deep-learning-generated species classifications to their iSpot account. The user can select which observations to include, as well as which classifications to include, and then we post these to their account on iSpot. In order to delegate this functionality (and this functionality only) to FASTCAT-Cloud the user must add the user ID of their iSpot account to their FASTCAT-Cloud account. This is an integration that the iSpot team has developed alongside us, as this functionality did not exist with them prior to this.

We plan to develop integrations with other project partners in the coming months, for example through upload mechanisms with STAplus.

3.5 Walk-through with examples

1. User goes to the website: https://service.fastcat-cloud.org/



Automatic species detection

FASTCAT-Cloud utilises state of the art deep learning algorithms to automatically identify all animals in your images and identify their species

Remove Humans

Storing images and videos of humans captured accidentally by your camera trap poses privacy concerns. Our pipeline automatically removes any images containing humans so that you don't have to.

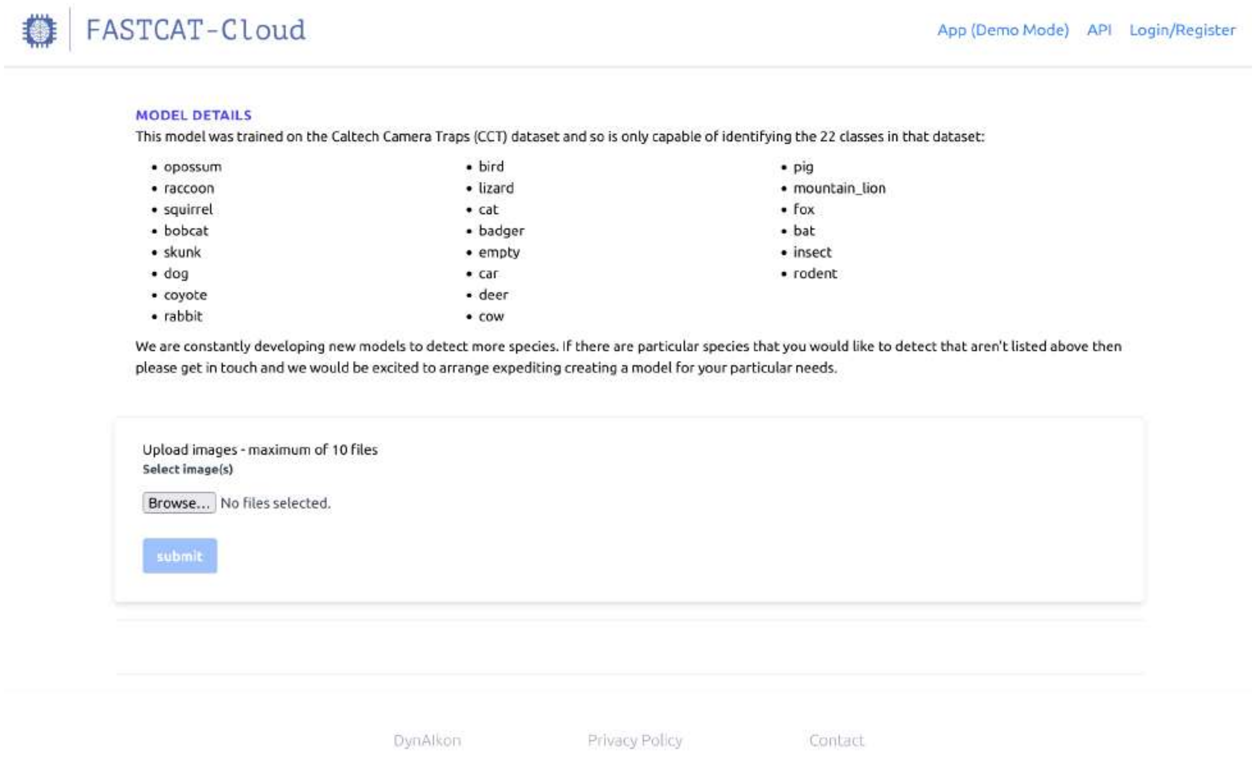
Detect Species

Manually annotating all of your camera trap images is a tedious task. Our pipeline automates this process so that you don't have to. As well as that, we provide the location of the animal in the image and draw a box around it so you can't miss it!

Summarise Observations

Want to know how many observations of bobcats you have in total? Our tool provides an easy-to-digest summary of the observations uploaded.

2. User navigates to the main app page using the navigation in the website header: https://service.fastcat-cloud.org/app



MODEL DETAILS

This model was trained on the Caltech Camera Traps (CCT) dataset and so is only capable of identifying the 22 classes in that dataset:

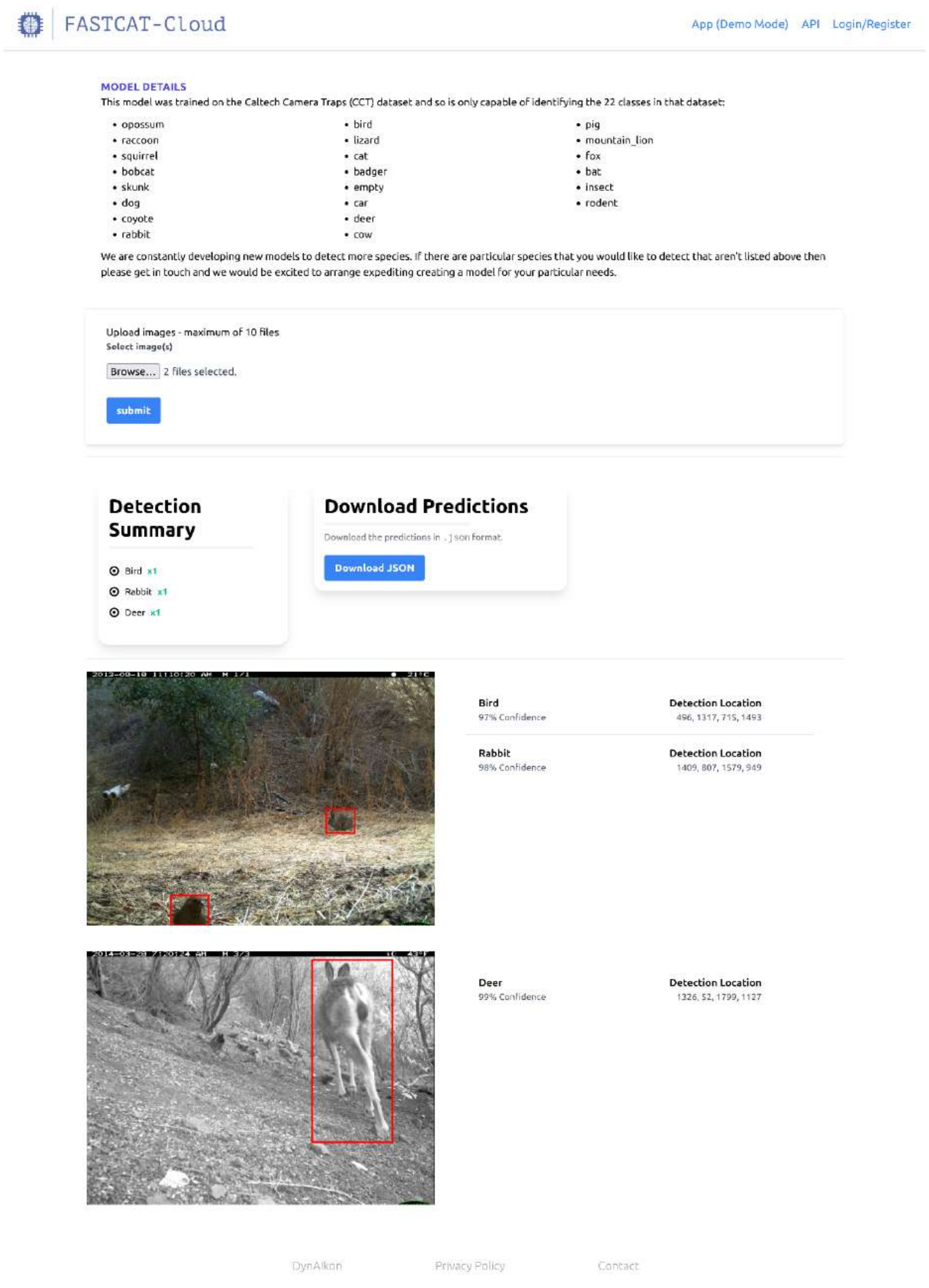
- opossum
- raccoon
- squirrel
- bobcat
- skunk
- dog
- coyote
- rabbit
- bird
- lizard
- cat
- badger
- empty
- car
- deer
- cow
- pig
- mountain_lion
- fox
- bat
- insect
- rodent

We are constantly developing new models to detect more species. If there are particular species that you would like to detect that aren't listed above then please get in touch and we would be excited to arrange expediting creating a model for your particular needs.

Upload images - maximum of 10 files

Select image(s)

3. User selects (uploads) images on their computer, and then clicks the submit button.



MODEL DETAILS

This model was trained on the Caltech Camera Traps (CCT) dataset and so is only capable of identifying the 22 classes in that dataset:

- opossum
- raccoon
- squirrel
- bobcat
- skunk
- dog
- coyote
- rabbit
- bird
- lizard
- cat
- badger
- empty
- car
- deer
- cow
- pig
- mountain_lion
- fox
- bat
- insect
- rodent

We are constantly developing new models to detect more species. If there are particular species that you would like to detect that aren't listed above then please get in touch and we would be excited to arrange expediting creating a model for your particular needs.

Upload images - maximum of 10 files

Select image(s)

Browse... 2 files selected.

submit

Detection Summary

- Bird x1
- Rabbit x1
- Deer x1

Download Predictions

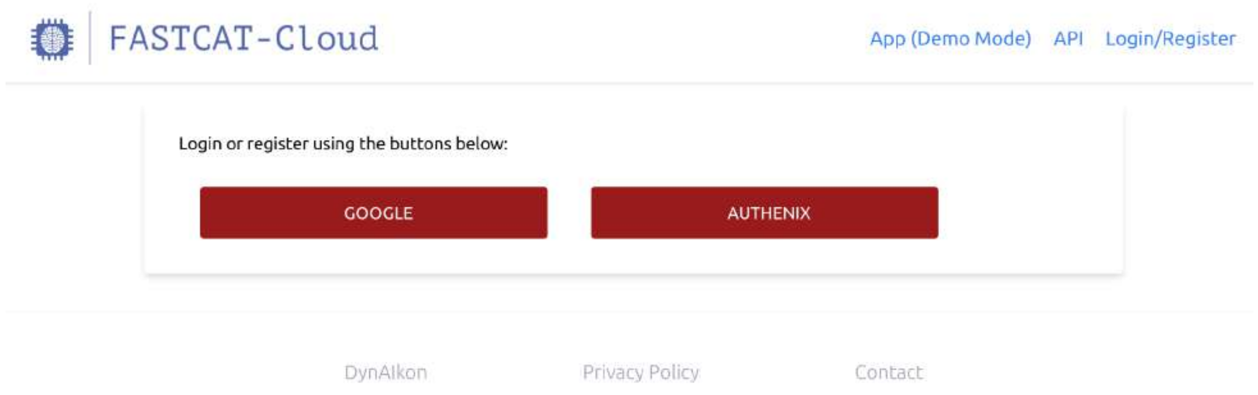
Download the predictions in .json format.

Download JSON

Bird 97% Confidence	**Detection Location** 496, 1317, 715, 1493
Rabbit 98% Confidence	**Detection Location** 1409, 807, 1579, 949
Deer 99% Confidence	**Detection Location** 1326, 52, 1799, 1127

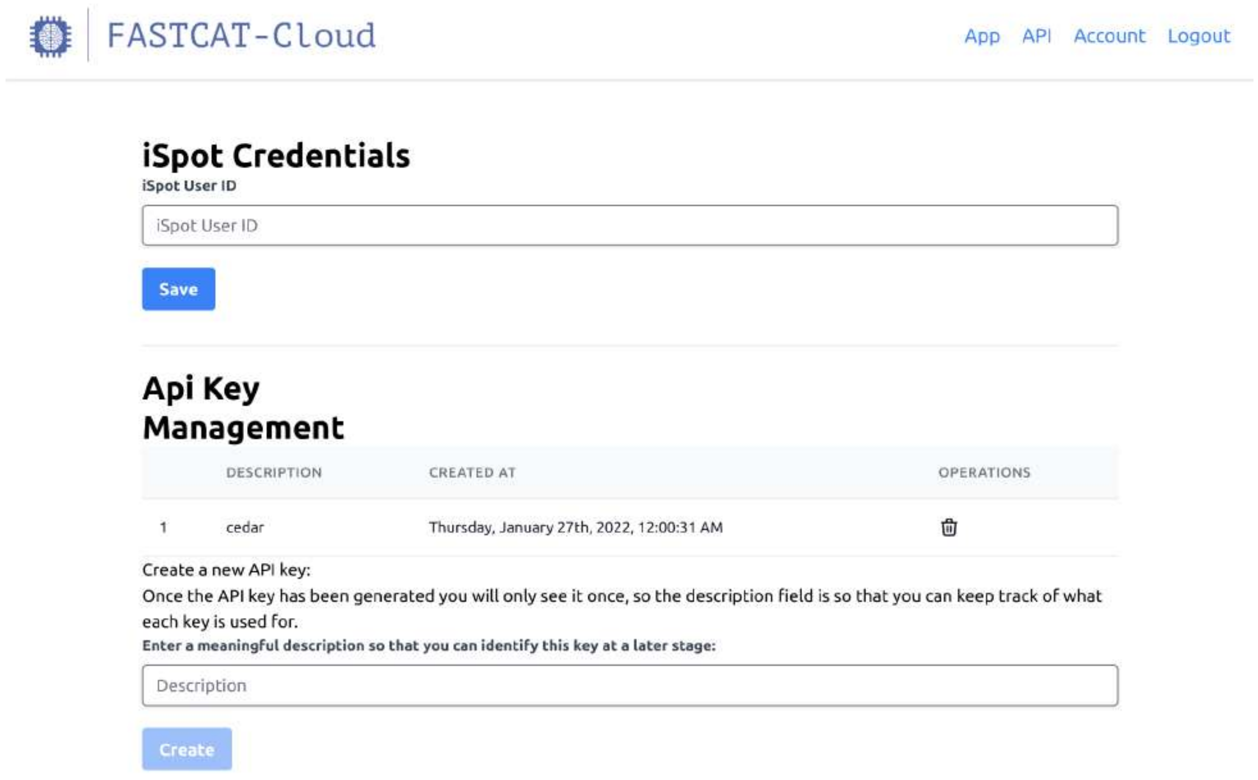
As can be seen above the images that the user submitted are displayed on the webpage, and the results from the FASTCAT-Cloud API are returned to the user and displayed alongside the images. In addition, the bounding boxes produced by the FASTCAT-Cloud API are drawn onto the images to make it easier to locate the animals. Above the images the user is presented with a summary of observations as well as a button to download the results in JSON format for convenience or integration into their other data processing needs.

4. User navigates to the Account page: https://service.fastcat-cloud.org/account



Login or register using the buttons below:

5. User sees two options to login, either with Google or Authenix. User logs in with an option.



iSpot Credentials

iSpot User ID

iSpot User ID

Api Key Management

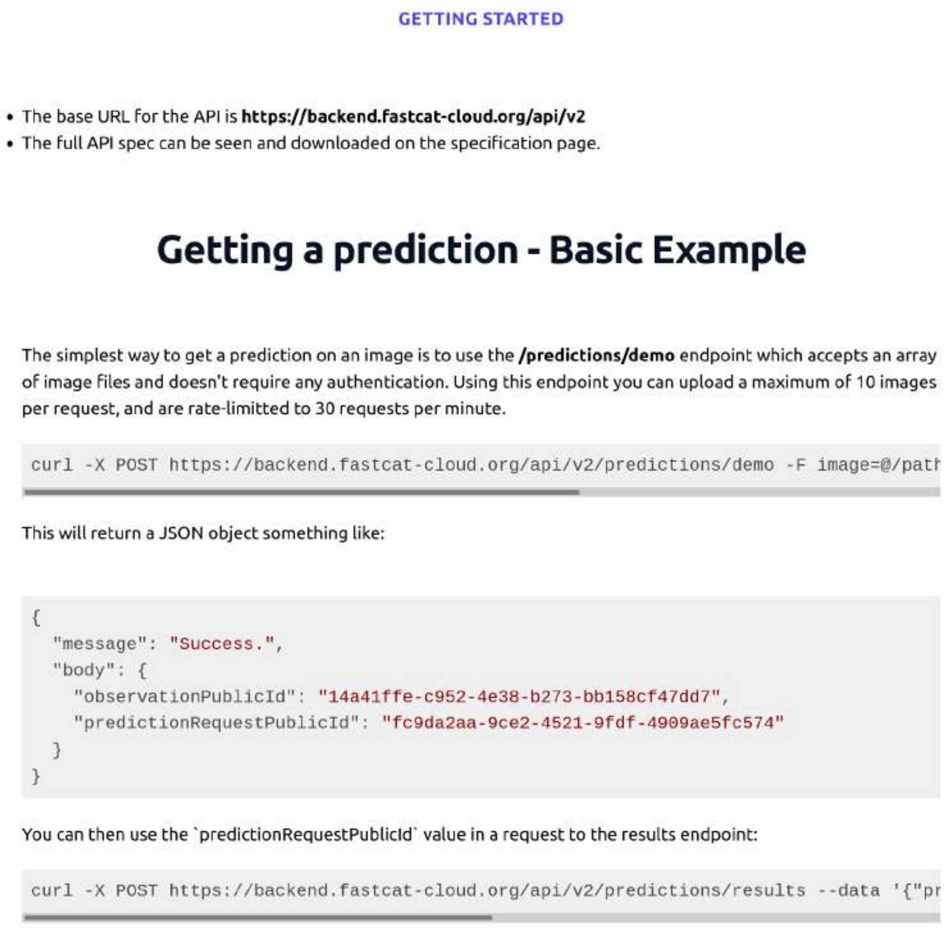
	DESCRIPTION	CREATED AT	OPERATIONS
1	cedar	Thursday, January 27th, 2022, 12:00:31 AM	🗑

Create a new API key:
Once the API key has been generated you will only see it once, so the description field is so that you can keep track of what each key is used for.
Enter a meaningful description so that you can identify this key at a later stage:

Description

After the user has logged in they can see options to manage API keys associated with their account. These keys can be used when authenticating with the API programmatically such as through a data processing script or through a smart camera trap such as our own DynAikonTrap/FASTCAT-Edge.

6. The user wants to use the FASTCAT-Cloud API directly and so they navigate to the API page: https://service.fastcat-cloud.org/api . User sees getting started examples.



GETTING STARTED

- The base URL for the API is **https://backend.fastcat-cloud.org/api/v2**
- The full API spec can be seen and downloaded on the specification page.

Getting a prediction - Basic Example

The simplest way to get a prediction on an image is to use the **/predictions/demo** endpoint which accepts an array of image files and doesn't require any authentication. Using this endpoint you can upload a maximum of 10 images per request, and are rate-limitted to 30 requests per minute.

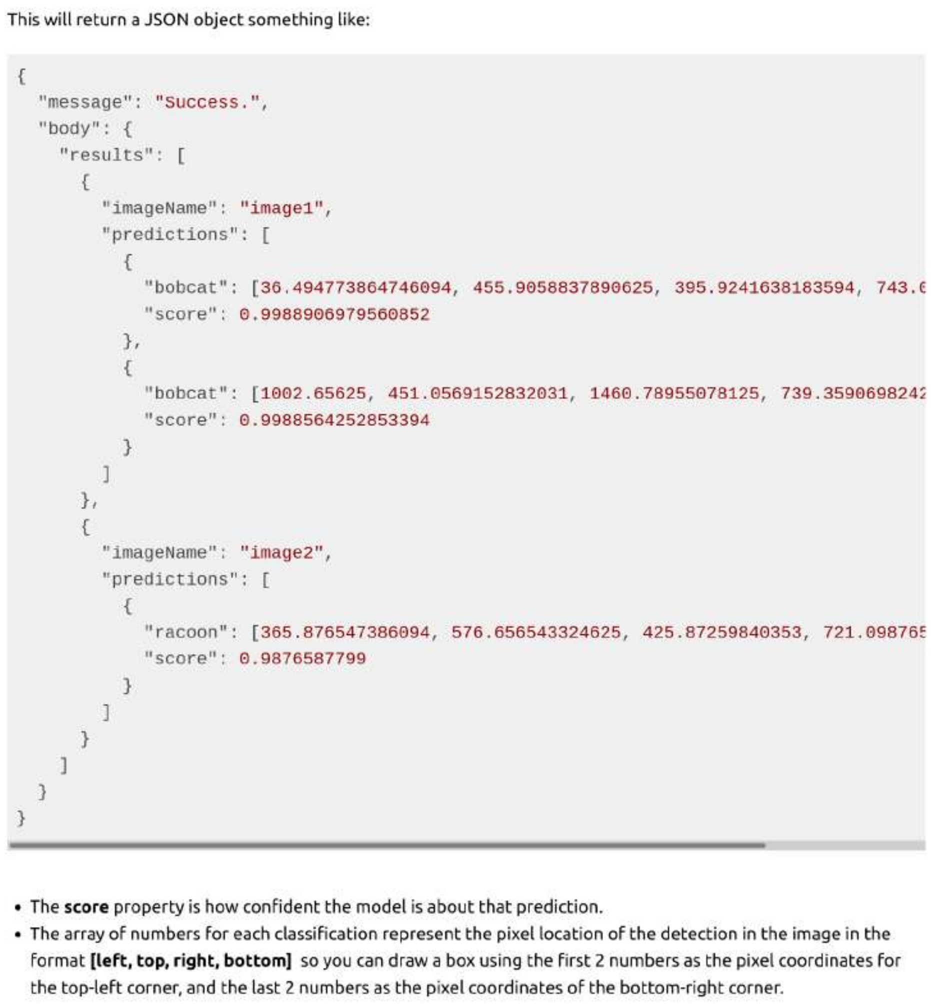
```
curl -X POST https://backend.fastcat-cloud.org/api/v2/predictions/demo -F image=@/path
```

This will return a JSON object something like:

```
{
  "message": "Success.",
  "body": {
    "observationPublicId": "14a41ffe-c952-4e38-b273-bb158cf47dd7",
    "predictionRequestPublicId": "fc9da2aa-9ce2-4521-9fdf-4909ae5fc574"
  }
}
```

You can then use the `predictionRequestPublicId` value in a request to the results endpoint:

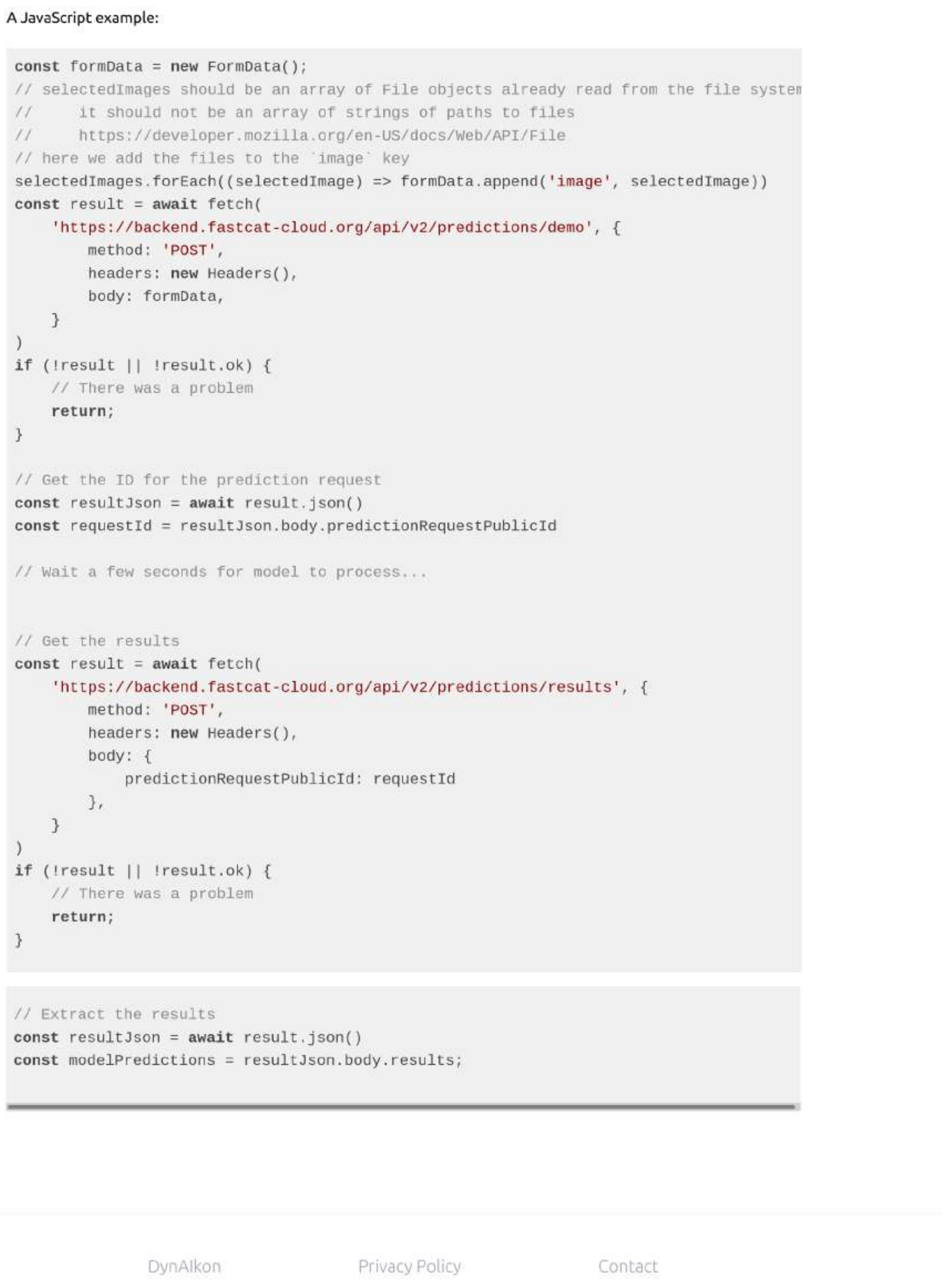
```
curl -X POST https://backend.fastcat-cloud.org/api/v2/predictions/results --data '{"pr
```



This will return a JSON object something like:

```
{
  "message": "Success.",
  "body": {
    "results": [
      {
        "imageName": "image1",
        "predictions": [
          {
            "bobcat": [36.494773864746094, 455.9058837890625, 395.9241638183594, 743.6
            "score": 0.9988906979560852
          },
          {
            "bobcat": [1002.65625, 451.0569152832031, 1460.78955078125, 739.3590698242
            "score": 0.9988564252853394
          }
        ]
      },
      {
        "imageName": "image2",
        "predictions": [
          {
            "racoon": [365.876547386094, 576.656543324625, 425.87259840353, 721.098765
            "score": 0.9876587799
          }
        ]
      }
    ]
  }
}
```

- The **score** property is how confident the model is about that prediction.
- The array of numbers for each classification represent the pixel location of the detection in the image in the format **[left, top, right, bottom]** so you can draw a box using the first 2 numbers as the pixel coordinates for the top-left corner, and the last 2 numbers as the pixel coordinates of the bottom-right corner.



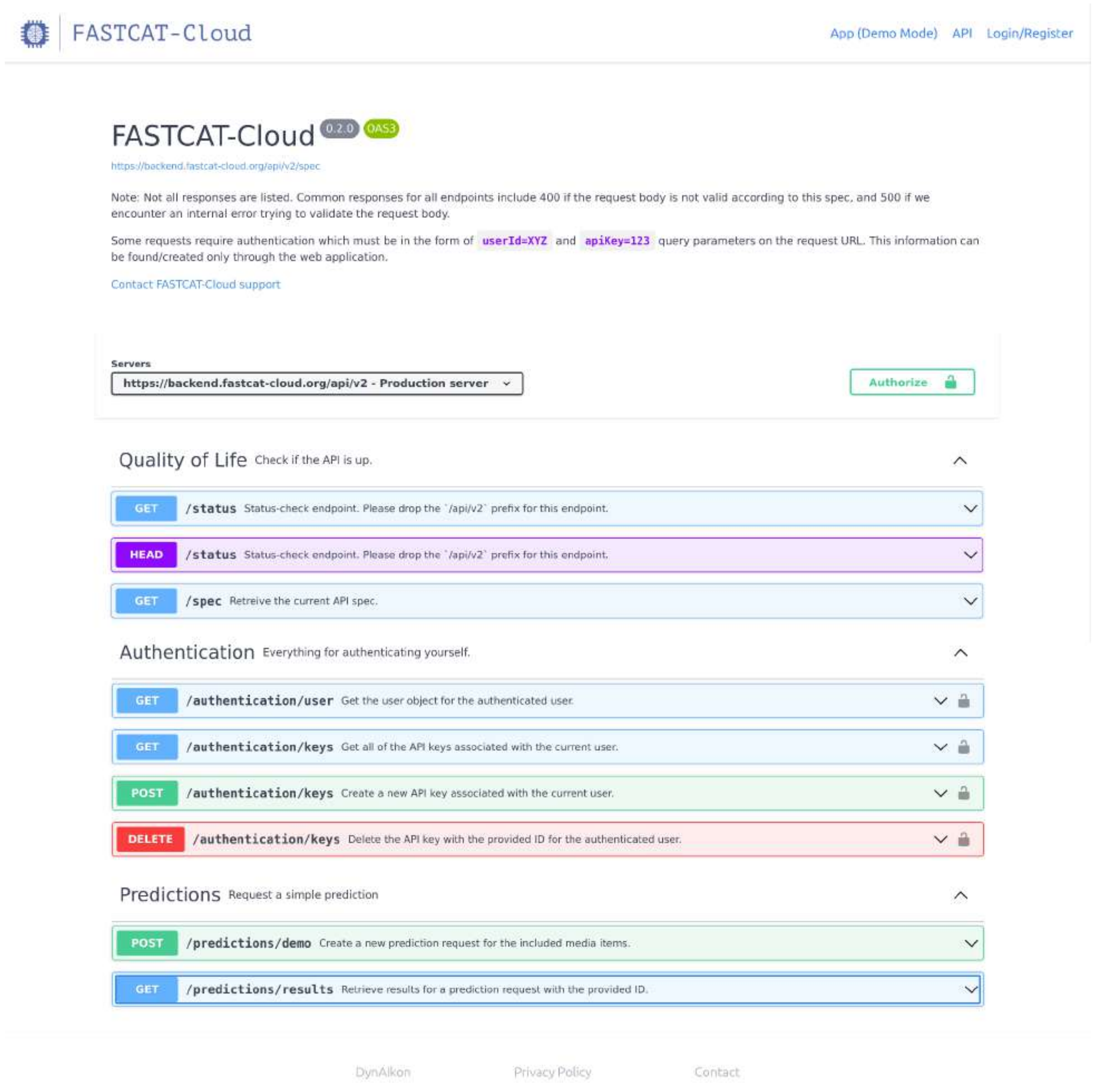
A JavaScript example:

```
const formData = new FormData();
// selectedImages should be an array of File objects already read from the file system
//     it should not be an array of strings of paths to files
//     https://developer.mozilla.org/en-US/docs/Web/API/File
// here we add the files to the `image` key
selectedImages.forEach((selectedImage) => formData.append('image', selectedImage))
const result = await fetch(
    'https://backend.fastcat-cloud.org/api/v2/predictions/demo', {
        method: 'POST',
        headers: new Headers(),
        body: formData,
    }
)
if (!result || !result.ok) {
    // There was a problem
    return;
}

// Get the ID for the prediction request
const resultJson = await result.json()
const requestId = resultJson.body.predictionRequestPublicId

// Wait a few seconds for model to process...


// Get the results
const result = await fetch(
    'https://backend.fastcat-cloud.org/api/v2/predictions/results', {
        method: 'POST',
        headers: new Headers(),
        body: {
            predictionRequestPublicId: requestId
        },
    }
)
if (!result || !result.ok) {
    // There was a problem
    return;
}
```

```
// Extract the results
const resultJson = await result.json()
const modelPredictions = resultJson.body.results;
```

DynAIkon   Privacy Policy   Contact

7. The user wants detailed technical information about the API and so they navigate to the specification page: https://service.fastcat-cloud.org/api/spec



FASTCAT-Cloud 0.2.0 OAS3

https://backend.fastcat-cloud.org/api/v2/spec

Note: Not all responses are listed. Common responses for all endpoints include 400 if the request body is not valid according to this spec, and 500 if we encounter an internal error trying to validate the request body.

Some requests require authentication which must be in the form of `userId=XYZ` and `apiKey=123` query parameters on the request URL. This information can be found/created only through the web application.

Contact FASTCAT-Cloud support

Servers
https://backend.fastcat-cloud.org/api/v2 - Production server

Authorize

Quality of Life Check if the API is up.

GET /status Status-check endpoint. Please drop the `/api/v2` prefix for this endpoint.

HEAD /status Status-check endpoint. Please drop the `/api/v2` prefix for this endpoint.

GET /spec Retreive the current API spec.

Authentication Everything for authenticating yourself.

GET /authentication/user Get the user object for the authenticated user.

GET /authentication/keys Get all of the API keys associated with the current user.

POST /authentication/keys Create a new API key associated with the current user.

DELETE /authentication/keys Delete the API key with the provided ID for the authenticated user.

Predictions Request a simple prediction

POST /predictions/demo Create a new prediction request for the included media items.

GET /predictions/results Retrieve results for a prediction request with the provided ID.

Conclusion and perspectives

Although we have achieved with this deliverable our initial objectives, we have plans to keep improving the potential of the FASTCAT-Cloud service.

In the next phase, we will provide a larger range of different models that are capable of detecting different sets of species, with models that are well suited for different geographical regions, models that are well suited to specific species families, or models that are particularly good at processing given specific environmental conditions, such as night-time, or under-water capture of videos.

We will also keep increasing the number of integrations with Citizen Observatories. In particular we are in the process of getting our service offered on the EOSC platform.

Glossary

Abbreviation / key	Description
AI	Artificial Intelligence
API	Application Programming Interface. A set of defined methods that allows programs to interact with other programs.
Authenix	An authentication provider created by another part/dyn/projects/cos4cloud/d4.4/caltech-subset of this Cos4Cloud project.
AWS	Amazon Web Services
EC2	Amazon Elastic Compute Cloud
Bbox	Bounding box. A rectangular area describing the location of something in an image or video.
CCT	Caltech Camera Traps dataset
CNN	Convolutional Neural Network
CO	Citizen Observatory
FASTCAT-Cloud	Flexible AI SysTem for CAmera Traps in the Cloud
FASTCAT-Edge	Flexible AI SysTem for CAmera Traps on the camera trap itself
GBIF	Global Biodiversity Information Facility. A large collection/database of species observations.
IoU	Intersection over Union. A metric used to measure the quality of a predicted bounding box compared to ground-truth data. The area of intersection of the two boxes divided by the area of the union of the two boxes.
JSON	A standard file format commonly used in programming.
mAP	Mean Average Precision. A metric used to measure the quality of a bounding box and corresponding classification prediction. mAP^{50} and mAP^{75} refer to the 50% and 75% threshold variants of the metric.
MS COCO	Common Objects in Context dataset
NN	Neural Network

NodeJS	A server-side (or general backend) JavaScript runtime environment
OAuth2	An authorization/authentication protocol
OpenAPI Specification	A popular syntax for describing RESTful APIs. Previously known as a Swagger specification
STAplus	SensorThingsAPI plus (an extension of STA by the consortium)
Swagger Specification	The old name for an OpenAPI specification
TRL	Technology readiness level
WCS	Wildlife Conservation Society dataset
YAML	A data serialisation language that is commonly used for configuration files as they are easily readable by humans. Other examples of data serialisation languages include TEXT, HTML, PHP, JS, C, etc.

References

- Beery, Sara, Grant Van Horn, and Pietro Perona: Recognition in terra incognita. Proceedings of the European Conference on Computer Vision (ECCV). 2018.
- Beery, Sara, Dan Morris, and Siyu Yang: Efficient pipeline for camera trap image review. arXiv preprint arXiv:1907.06772 (2019).
- Cai, Zhaowei, and Nuno Vasconcelos. Cascade R-CNN: High Quality Object Detection and Instance Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence. (2019): 1–1.
- Chen, Kai, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin: MMDetection: Open MMLab Detection Toolbox and Benchmark. arXiv preprint arXiv:1906.07155. (2019).
- Dai, Jifeng, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei: Deformable Convolutional Networks. Proceedings of the IEEE International Conference on Computer Vision. (2017).
- Everingham, Mark, Luk Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman: The PASCAL visual object classes (VOC) challenge. International Journal of Computer Vision, 88 (2), 303–338. (2010).
- Girshick, Ross: Fast R-CNN. Proceedings of the IEEE international Conference on Computer Vision. (2015).
- Girshick, Ross, et al: Detectron. https://github.com/facebookresearch/detectron. (2018).
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun: Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016).
- Li, Xiang, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, Jian Yang: Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection. Advances in Neural Information Processing Systems 33. (2020).
- Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick: Microsoft COCO: Common objects in context. European Conference on Computer Vision. Springer LNCS 8693. (2014).
- Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár: Focal loss for dense object detection. Proceedings of the IEEE International Conference on Computer Vision. (2017a).
- Lin, Tsung-Yi, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie: Feature pyramid networks for object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017b).

- Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg: SSD: Single Shot MultiBox Detector. European Conference on Computer Vision. Springer LNCS 9905. (2016).
- MMCV Contributors: MMCV: OpenMMLab Computer Vision Foundation. https://github.com/open-mmlab/mmcv. (2018).
- Petso, Tinao, Rodrigo S. Jamisola, and Dimane Mpoeleng: Review on methods used for wildlife species and individual identification. European Journal of Wildlife Research 68.1 (2022): 1-18.
- Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun: Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in Neural Information Processing Systems 28 (2015): 91-99.
- Swanson, Alexandra, Margaret Kosmala, Chris Lintott, Robert Simpson, Arfon Smith, and Craig Packer: Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna. Scientific Data volume 2, Article number: 150026. (2015).
- Wang, Jiaqi, Kai Chen, Shuo Yang, Chen Change Loy, and Dahua Lin: Region Proposal by Guided Anchoring. IEEE Conference on Computer Vision and Pattern Recognition. (2019).
- Wu, Yuxin, et al: Detectron2. https://github.com/facebookresearch/detectron2. (2019).
- Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He: Aggregated residual transformations for deep neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017).
- Yang, Ze, Shaohui Liu, Han Hu, Liwei Wang, and Stephen Lin: Reppoints: Point set representation for object detection. Proceedings of the IEEE/CVF International Conference on Computer Vision. (2019).
- Zhu, Xizhou, Han Hu, Stephen Lin, and Jifeng Dai: Deformable ConvNets v2: More Deformable, Better Results. IEEE/CVF Conference on Computer Vision and Pattern Recognition. (2019).
- Zhu, Xizhou, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai: Deformable DETR: Deformable Transformers for End-to-End Object Detection. International Conference on Learning Representations. (2020).
- Zhang, Hang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola: ResNeSt: Split-Attention Networks. arXiv preprint arXiv:2004.08955. (2020).

Annex 1 - Understanding Model Names

ResNet model variations are indicated with the format *A-B* where *A* is a letter representing the architectural variant and *B* is a number indicating the number of layers from the model used.

Architecture	**Letter used to represent the architecture**
ResNet (He, 2016)	R
ResNeXt (Xie, 2017)	X
ResNeSt (Zhang, 2020)	S

Examples:

- R-50: uses the ResNet architecture with 50 layers
- X-101: uses the ResNeXt architecture with 101 layers
- S-101 uses the ResNeSt architecture with 101 layers

Further to this ResNeXt architectures offer a further variation given in the form *C*x*D*d where *C* is the cardinality parameter of the network and *D* is the width of the network. A deeper explanation is provided in Sections 3.1 and 5.1 of the architecture paper (Xie et al., 2017). An example of this would look like *X-101 64x4d*.

Model names including *FPN* indicate that they make use of a feature pyramid in their early layers as per Lin (Lin et al, 2017b).

Model names including DCN indicate that they make use of the deformable technique described by Zhu et al. (2019).