ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
ENVIRONMENTAL COMPUTATIONAL SCIENCE AND EARTH OBSERVATION LABORATORY

Automated monitoring of insects

MASTER THESIS

Léonard PASI

QUEBEC ARTIFICIAL INTELLIGENCE INSTITUTE

Supervisors:

Prof. Devis Tuia (EPFL)
Prof. David Rolnick (MILA)

September 1, 2023

Contents

Chapter 1

Introduction

The history of *Homo Sapiens* as an ecological killer begins earlier than many people imagine. Around 50,000 years ago, Homo Sapiens first set foot in Australia; within a few thousand years, of the twenty-four genera of Australian land animals weighing fifty kilograms or more, twenty-three became extinct [1]. These included *Thylacoleo carnifex*, a marsupial lion that was the continent's largest predator; *Genyornis newtoni*, a flightless bird over two meters in height; and *Procoptodon goliah*, a giant kangaroo weighing over 200 kg. The colonization of the American continent led to a similar ecological catastrophe. *Homo Sapiens* started migrating south from Alaska about 14,000 years ago. In less than two millennia, humans had reached the continent's southern tip. Concurrently, 72% and 83% of megafaunal genera become extinct in North and South America, respectively [2]. While the relative importance between (natural) climate change and humans is still debated, the latest research suggests that the latter were the principal or necessary driver of the major extinction events in Australia and South America [3, 4].

Today, the biosphere is undergoing its 6th mass extinction [5, 6]. A mass extinction occurs when the Earth loses more than three-quarters of its species in a geologically short interval [7]; the last one marked the end of the dinosaurs' era, 66 million years ago, and was caused by a 10km-wide asteroid colliding with planet Earth. In contrast with all the previous, today's mass extinction is mostly driven by one species, *Homo Sapiens*. In fact, humans are responsible for all the major immediate causes of biotic destruction: habitat conversion, climate disruption, overexploitation, toxification, species invasions and disease [5]. The extinctions span numerous species of plants [8] and animals, including mammals, birds, reptiles, amphibians, fish, and insects [9].

Insects are the most diverse group of animals. They include more than a million described species and represent more than half of all known living organisms [10]; moreover, it is estimated that 80% of insect species are still to be discovered [11]. Recent studies have demonstrated alarming rates of insect diversity and abundance loss [12, 13, 14]. However, the data on changes in insect species diversity and abundance has substantial taxonomic, spatial, and temporal biases and gaps [15]. A major reason for these short-falls is the inherent difficulty of identifying insects: expert knowledge is necessary to classify all insects collected in a trap, which makes this approach time- or cost-prohibitive to scale, especially as insect identification expertise is in decline [16]. Using indicator species can be an effective approach to sidestep this problem, but doing so often results in inadequate knowledge and compromised measures of interest [17].

In the last decade, another approach has emerged in species monitoring studies in order to drastically reduce the dependence on manual labor by automatically processing large amounts of collected data: the use of deep learning. As part of this trend, in 2021, a team of researchers in Denmark published their work on a novel, automatic light trap to monitor moths using computer vision-based tracking and deep learning [18]. Moths make up the vast majority of the order *Lepidoptera*, which by itself accounts for around 10% of all described species of living organisms. Moths are important as pollinators, herbivores and prey; as such, changes in the abundance of moths could have cascading effects through the food web. Additionally, some of the most damaging pest species in agriculture and forestry are also moths [19, 20], suggesting they are a very relevant group of insects to monitor more effectively.

As depicted in Fig.1.1, the system presented in [18] consists of a UV light to attract live moths during night hours, a backlit white screen for the moths to rest on, a high-resolution web camera with a light ring, a computer and a powered junction box with DC-DC converter. A sequence of images is captured and stored on a hard drive whenever a change within the camera field of view is detected by the computer vision system. On warm summer nights with a high level of insect activity, more than 20,000 images are captured per night. The images are processed off-line through a pipeline that involves four steps. First, object detection is performed with classic computer vision techniques: Otsu's method to separate foreground from background, morphological operations to filter out small noisy blobs and close blobs, and connected-component labeling. Second, tracking is used to ensure that each insect is only counted once during its stay in the camera field of view. Third, each insect track is classified through a CNN into ten different classes, representing frequently observed species, groups of very similar species, or false object detections without insects. Fourth, a summary of the individuals detected and tracked by the algorithm is derived.

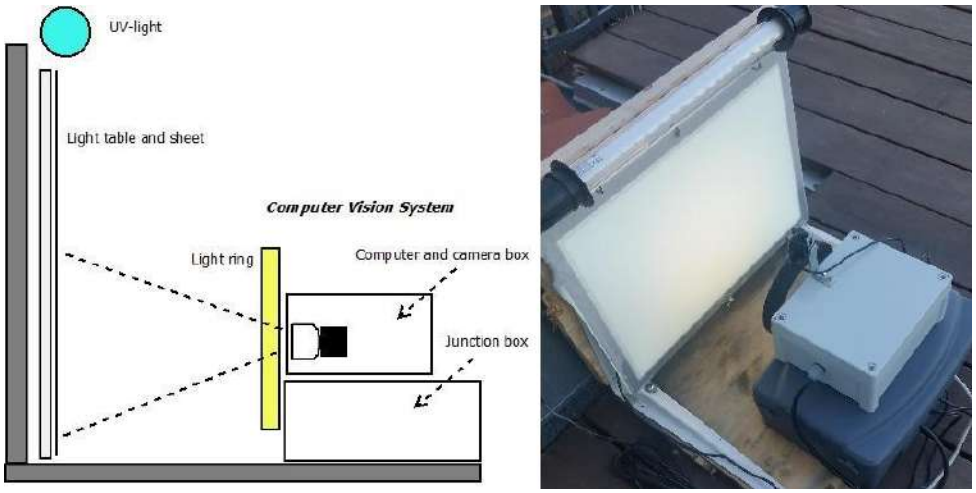


Figure 1.1: The portable light trap with a light table, a white sheet, and UV light to attract live moths during night hours. The computer vision system consists of a light ring, a camera with a computer and electronics, and a powered junction box with DC-DC converter. From [18].

Naturally, moths that fly in and out of the camera field of view will be counted multiple times; however, it is noted that moths tend to be rather stationary once they land on the sheet. Ad-

ditionally, supposing that the average number of visits per individual is fixed over time, this doesn't prevent from observing trends in species abundance. Hence, the system should be considered as a viable alternative to traditional methods that typically require tedious manual labor (i.e., visiting the trap several times in a season for observation) and often result in the killing of rare species of insects. Nonetheless, there is vast room for improvement. Around the same time the research in [18] was published, prof. David Rolnick got involved with the project, and created a team at Mila—of which I was part for this thesis—to work on this challenge. In the following paragraphs, I give a brief overview of the main limitations of the system as presented in [18], as well as the corresponding solution developed by the team at Mila.

One major difficulty in the project was to obtain enough training data for the classifier, especially as some of the target species had few occurrences. As the team in [18] resolved to have a balanced dataset, only 250 images per class were selected. Using extensive data augmentation, the dataset was scaled up by a factor of 32, and a ResNet-50 [21] model with pretrained weights achieved high accuracy on the validation set[1]. However, the performance of the model after deployment dropped, for multiple reasons.

One reason was the presence of insect species outside the training dataset. A training dataset including all possible species in the region would be necessary, but given the huge diversity of insects, it would be impossible for a small team of researchers to build such a dataset from scratch. The solution proposed by the team at Mila is to use images from iNaturalist and Observation.org, accessible through the Global Biodiversity Information Facility (GBIF). Given a list of moth species (typically multiple thousands) relevant in a certain region, datasets of hundred thousand labelled images can be created and used to train the model. Furthermore, the classification task was decomposed in two sub-tasks: first, moth/non-moth binary classification; second, fine-grained species classification for moths. Both classifiers are trained on GBIF data.

Other significant sources of errors in the system described in [18] were the object detection and the tracking algorithm. The team at Mila improved both these steps: first, the classical computer vision techniques were replaced by a deep learning model for object detection; second, the tracking algorithm was enhanced by introducing the similarity in the classifier's feature space to the similarity metric between detections in consecutive images. The new pipeline is summarized in Fig.1.2.

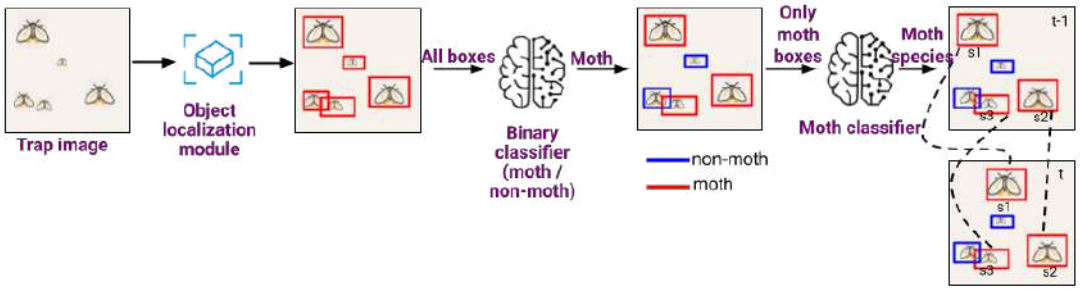


Figure 1.2: Current pipeline.

[1]There was perhaps a methodological error in the model evaluation. To capture the model performance on new data, the split between training and validation datasets should have been done before augmentation.

From these exciting innovations arise new exciting challenges. With an inference time of more than 3 seconds on CPU, the object detector is too slow, and its accuracy can still largely be improved; the work to address these shortcomings will be presented in chapter 2. The new classifier is affected by the domain shift between the GBIF training data and the target data from moth traps. While strong data augmentation operations have mitigated the problem, even better results are expected if the GBIF training datasets can be enhanced with manually labeled images from the traps. The implementation of active learning techniques to make the most of the manual work will be discussed in chapter 3.

Chapter 2

Object Detection

When I joined the team in May, the speed of the object detector was a point of concern. For the collaborators and users of the project without GPUs, the model's inference time of more than three seconds on CPU was a big limitation. Additionally, there was an idea to transfer the object detection step of the pipeline to the deployments, in order to reduce the amount of stored data. To keep the costs down, the deployments would also not be equipped with GPUs. For these two reasons, increasing the object detector speed was very attractive.

The object detector was a Faster R-CNN with a ResNet-50-FPN backbone [22, 23], as implemented in PyTorch's torchvision package. Its goal is to localize insects. It was trained on a set of images on which bounding boxes could accurately be inferred with classical computer vision techniques, due to the low density of moths and clean background. My efforts to improve the model can be grouped around three main axes: the framework, the training data, and the architecture.

2.1. The framework

Having a good framework is fundamental to allow efficient development of a model. One needs to be able to easily (i) keep track of experiments and link checkpoints to training runs, (ii) integrate new functionalities, and (iii) evaluate the models. As the work on the classification part of the pipeline had taken most of the team's time, none of this was in place for the object detection.

2.1.1. Code improvements and experiment tracking

The code was organized in a number of Jupyter Notebooks and some python modules. Jupyter Notebooks are fine for early data analysis and exploration, but they certainly have little place in a more advanced project. The non-linear workflow and the lack of IDE features (such as linting and code styling warnings) favor errors and bad coding practices; additionally, they are terrible for code versioning[1], all reasons that make it hard for teammates to collaborate and expand the project. Hence, a full refactor was needed. The result was a few modules for dataset declarations and common functions definitions, and two command-line scripts, one for launching trainings, and one for launching inferences. The supported models are all those available in torchvision. The highest standards for code quality were held.

[1] This was less of a concern early on, as the team was not using a common repository

During development, tens of trainings can be launched in short intervals of time, and the number of models quickly start to add up. It is important to be able to link each model to a training run, with all its defining parameters. To this end, the popular Weights & Biases platform was adopted. The training configuration and the model weights are automatically uploaded to the platform; the model checkpoint is also saved locally, under a name that includes the model architecture and the training run ID assigned by W&B.

2.1.2. Model evaluation, metrics and threshold analysis

Visual inspection

The only way to evaluate the model was to run inferences and visualize the predicted bounding boxes. These were saved directly on copies of the images, which presents two major inconveniences: (i) it is very inefficient in terms of memory, and (ii) it makes it very unpractical to visualize the model performance as a function of the threshold. To address these issues, a simple python GUI application was created, using the standard tkinter package. A screenshot of the GUI is displayed in Fig.2.1. It is started from command line by running the python script, with two inputs: the path to the json file containing the model's predictions (which is outputted by the inference script), and, optionally, if they are in a different folder, the path to the images. The window presents: (i) a slider to modify the threshold (bounding boxes will appear and disappear accordingly); (ii) the path to the json file (a useful reminder when the app is run in parallel to compare models); (iii) the image filename; (iv) a counter indicating the current image number out of the total; (v) and an entry to modify the step (useful to quickly move across hundreds of images). The user moves from one image to the other with the left and right arrow keys. The GUI also accepts ground truths instead of model predictions, in which case the score threshold won't be displayed.

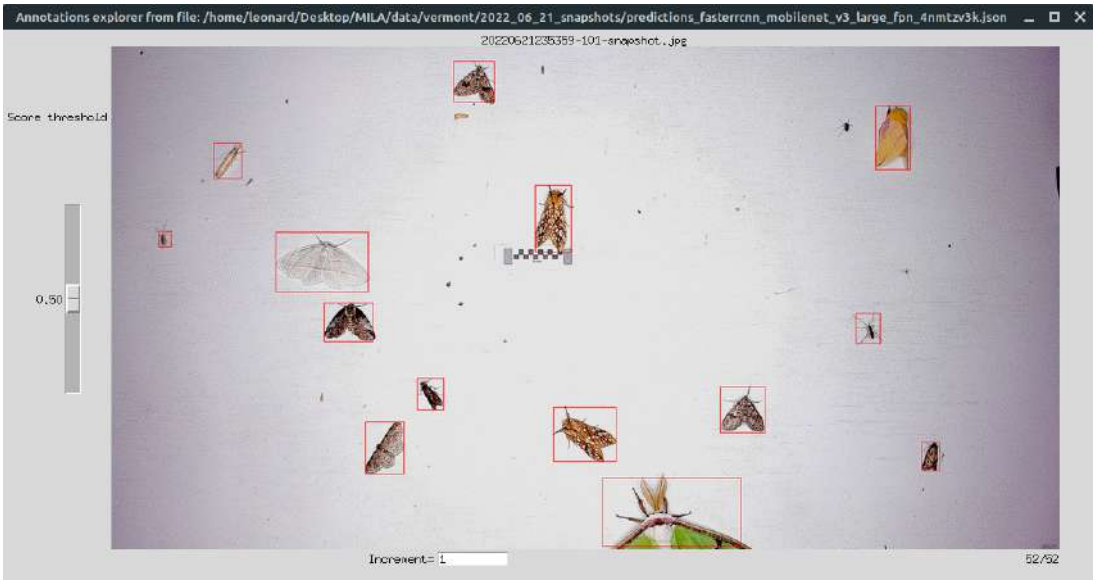


Figure 2.1: Screenshot of the GUI app to inspect models' predictions

Mean Average Precision

The GUI is a nice tool to inspect the model's predictions, however it is still a far cry from having actual testing datasets and metrics. In fact, visual inspection is both time-consuming and limited in its ability to measure small improvements, while also being affected by confirmation bias. With currently ongoing annotation efforts, the test sets will soon finally be available. In anticipation of that, I developed the relevant metrics. The mean average precision (mAP) is the standard metric for object detectors. In short, it is obtained by computing the average precision (AP) across of set of IoU (Intersection over Union) thresholds for each class and taking the mean of all APs. The AP is related to the area under the precision-recall (PR) curve, although it isn't exactly the same. In object detection, the PR curve depends on the IoU threshold, as it determines how strict the requirement is for a ground truth and a predicted bounding box to match (hence, it sets the boundary between true positives, false positives, etc.). For each IoU threshold, one PR curve and one AP measure is obtained. The mAP has the big advantage of being independent of the IoU threshold, which is important to evaluate models fairly. Its implementation in the torchmetrics library made it easy to integrate in the framework. During trainings, the mAP is computed on the evaluation set at each epoch and uploaded to W&B. Provided that the evaluation set is fixed, this offers a quick way to compare training on the W&B - much better than using the model's loss, which can change across models and is not interpretable.

Threshold analysis

While being a useful metric to compare models, the mAP doesn't help in setting the optimal threshold of the chosen model, which is a critical task. Not to be confused with the IoU threshold mentioned above, the score threshold determines whether a predicted bounding box is kept in the final prediction or not. It is therefore critical for the model's performance. A common way to choose the score threshold is from the precision recall curve. To make this operation possible, the computation of the PR curve was implemented from scratch, and a GUI was created. The GUI is presented in Fig.2.2. It takes as inputs the IoU threshold(s) at which to compute the PR curve, and the paths to the json files containing the ground truths and the model predictions. The user can set the model threshold and visualize the corresponding point on the PR curve, hence know what precision (i.e. the proportion of correct insect detections among all detections) and recall (i.e. the proportion of correct insect detections among all insects on the images) to expect from the model. Given a testing dataset for a specific deployment, this information would be very valuable for the ecologists using the system.

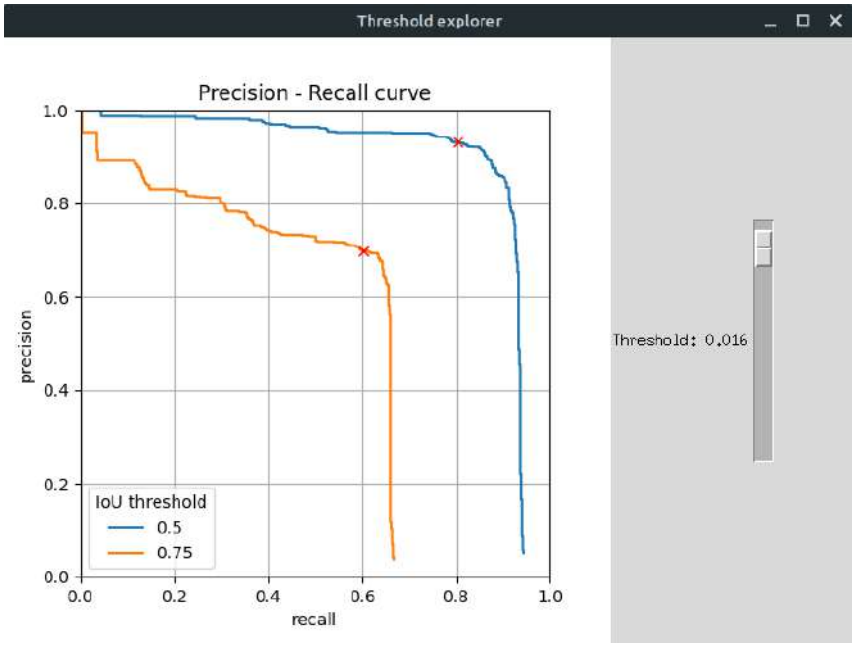


Figure 2.2: Precision-recall curve for threshold analysis

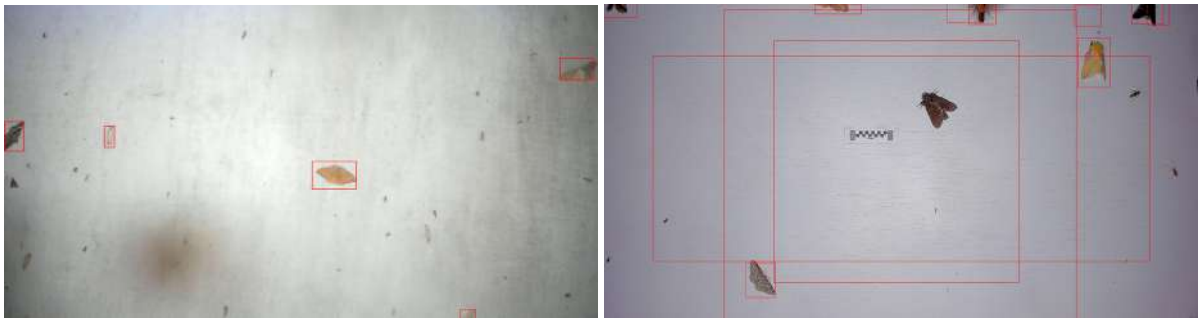
2.2. THE TRAINING DATA

2.2.1. The need for new training datasets

As mentioned before, the Faster R-CNN model was trained on a set of images on which bounding boxes could accurately be inferred with classical computer vision techniques, due to the low density of moths and clean background. A visual inspection of this dataset revealed that it is actually not from any of the current deployments[2], and that a different data acquisition technique was used, leading to way less sharp edges. This is a clear example of data-shift. In the past, quick tests with a SSDlite model —the lightest and faster architecture available on torchvision—, had failed. I hypothesized that one of the reasons could have been data-shift: while the relatively heavy Faster R-CNN with ResNet-50-FPN backbone can handle the shift, a smaller model is expected to have less generalization capacity, and hence be more sensible to it.

To test this hypothesis, a SSDlite model was trained. The validation set was created by selecting images after a certain time stamp, as opposed to a random split. A number of images were used as a gap between training and validation sets, and thus were unused. This ensured a certain diversity between validation and training images, as consecutive images can be very similar to one another. The model was tested on the validation set and on new images, and the predictions were inspected with the apposite GUI app. As shown in Fig.2.3, a significant drop in performance was observed between validation images and target images, confirming the initial hypothesis, and suggesting that a new training dataset was needed to facilitate the adoption of a light-weight model.

[2]Or at least, not from any of the deployments from which images available to the team at Mila are.



(a) Old image (b) Recent image

Figure 2.3: Performance comparison of a SSDlite model trained on old images. Reasonable predictions are given across a wide spectrum of thresholds on the old images (a), while absurd predictions are made on the new images (b). In the example, the largest square bounding box has the highest confidence.

2.2.2. Garbage in, garbage out

Following the findings described in the previous section, a dataset of two thousand images from four different deployments (Quebec, Vermont, Newfoundland and Panama) was assembled. Bounding boxes were derived automatically using the Faster R-CNN model. However, an extensive inspection of the predictions revealed that the model was prone to two types of errors: missed detections (i.e. false negatives), especially on smaller moths, and double detections (i.e. bounding boxes that group multiple moths together). Additionally, the model would occasionally also make multiple predictions on the same large moth. Finally, the bounding boxes were often too loose around the insects, which makes them not ideal to train on (object detection annotation best practices include drawing perfectly tight bounding boxes). These errors, some of which are displayed in Fig.2.4, reflect the deficiencies of the model's training data, where small moths are often not annotated, moths are seldom close to one another and bounding boxes are often loose.

Figure 2.4: Common mistakes from the old Faster R-CNN model.

The goal was no longer to make the object detector faster while maintaining the same level of accuracy: the accuracy needed to improve as well. Any model trained on the dataset with Faster R-CNN's bounding boxes would replicate the same mistakes (as the saying goes, "Garbage in, garbage out"). Hence, the efforts to create a new training dataset were not over.

2.2.3. Training data synthesis

One bold idea to create new training data was suggested by a teammate: use Meta's recently released Segment Anything Model (SAM, [24]) to crop insects, and paste the crops on empty background images; in other worlds, to synthesize the training dataset. An immediate concern was that a model trained on such a dataset would learn to detect pasted objects instead of actual insects, i.e. its performance would degrade on natural images. However, given the potential of the idea, the gamble was taken. The pipeline consists of three steps: (i) run the inferences with SAM, (ii) manually filter the crops, (iii) and synthesize new images.

Inferences with SAM

A python command-line script was developed to run the SAM model with custom parameters on the desired images. The script can be used to generate bounding boxes, which are saved in a json file, and crops of detected objects. In the first case, SAM is used as an object detector; unfortunately, while the predicted bounding boxes are perfectly tight, there are too many mistakes to consider SAM as a drop-in replacement of Faster R-CNN for the annotation of new datasets. Common errors include missed detections of small moths, and wings that are considered as separate objects; tweaking SAM's parameters to improve on one problem tended to make the other worse. For the following tasks, it was important not to miss the small moths, in order not to bias the final dataset towards larger moths. Hence, the parameters were tweaked accordingly.

A diverse collection of nearly 300 images was assembled from five different locations (Denmark, Vermont, Quebec, Panama and Newfoundland). The IDs[3] of the selected images were stored for reference. The image processing produced almost 4k crops. Each crop of a detected object consists of two arrays: one array is a crop of the image, the other is a boolean mask indicating the object inside the crop. Each array is saved in the npy format, NumPy's standard for persisting a single arbitrary NumPy array on disk. The .npy files are zipped in a .npz file, with a naming convention that allows to pair crops and their corresponding mask together when reading the file later on.

Manual review of the segmented objects

As stated above, SAM's predictions are not perfect. Hence, there is still need for manual review. Luckily, with the right tools, manual review is very fast: critically, it is *much* faster than drawing bounding boxes. This is because the only required operation is to delete wrong detections. All detections are perfectly tight around the object, so there is no need to adjust the bounding boxes. The tool developed for this task is a GUI application, presented in Fig.2.5. The user can swipe through the crops read from the .npz file with the left and right arrow keys. Bad crops can be deleted in a split second by pressing the delete key. When the first crop is deleted, the "Save as" and "Discard" buttons are activated, such that changes can be saved (under the desired filename) or discarded at any time. If the window is closed with unsaved changes, a

[3]Regrettably, there is actually no guarantee of uniqueness of the image filename across the whole project.

dialog box appears to ask whether to save the changes or not. The user can also play with the overlay that appears on the segmented object, by making it more or less transparent.

Figure 2.5: GUI app to review SAM's crops.

Image synthesis

The command line script developed to synthesize the new images has four main inputs: the .npz file with the crops, the path to a folder with background images, the number of new images to create from each background, and the number of crops to paste on each image. For each new image, moths are pasted one after the other at random locations. Collisions are taken care of: if a moth overlaps with one of the already pasted moths by more than a certain threshold[4], a new random position is selected. Similarly, moths are not allowed to be out of the image by more than a certain portion. Two types of simple augmentations are used: rotations (90°, 180°, 270°) and flips (horizontal and vertical). To maximize the diversity of the augmented crops, the transformations are applied cyclically: first, the moths are pasted without augmentations; when all the moths in the collection have been pasted once, the algorithm cycles again through the collection, and a first set of augmentations is applied; at the second cycle, another set is chosen, and so on. A synthetic image is displayed in Fig2.6 as an example.

With the GUI described before, a collection of 2600 clean crops was created, during a single day of work. From these, using a collection of more than a hundred empty background images, a dataset of 5k images with was synthesized. For comparison, manual annotation of that many images is expected to take at least a week. Most importantly, it appeared that the bet paid off: models trained on synthetic data performed well on natural data –better than the previous model. The results will be presented more in details in the corresponding section. Lastly, it is worth noting one advantage of this technique: whenever a new deployment is created,

[4]in terms of overlapping surface over moth area

corresponding background images can be used with the existing collection of segmented insects to update the training dataset; this way, models can quickly be made familiar with new deployments backgrounds, which might be beneficial for the models' accuracy.

Figure 2.6: A synthetic image. In the last version of the synthetic dataset, moth density was increased from twenty to thirty moths per image.

2.3. MODEL ARCHITECTURE AND TRAINING RECIPE

SSDlite

To improve the speed of the model, changes in the architecture are necessary. In torchvision, many object-detection models are available off-the-shelf. A very informative table gives an overview of the models, with their performance on the standard COCO benchmark dataset, their size, and a link to the training recipe. As stated before, the fastest available model —by far—- is SSDlite, an adaptation of the Single Shot Detector [25] which was first briefly introduced on the MobileNetV2 paper [26] and later reused on the MobileNetV3 paper [27]. Initially, efforts were pursued to obtain a SSDlite model with good performance. To do so, following the indicated training recipe, some functionalities were added to the training script: the cosine annealing with warm-up epochs learning rate schedule (see Fig. 2.7), and the random crops and random horizontal flips data augmentations. While each of these delivered improvements, the model performance –as measured on the validation sets and logged to W&B for each training run– remained unsatisfactory. More precisely, the models were too sensitive to the score threshold. When the synthetic datasets were introduced, the performance on the (synthetic) validation sets dropped. In part, this was revealed to be due to the decrease in bounding box size (caused by the tight fit), which is shown in Fig.2.8. Still, the idea to use SSDlite as implemented in torchvision was abandoned. In hindsight, this should have been done much before; SSDlite internally resizes the images to 320x320 pixels, so it is only normal that it performs poorly on such small objects.

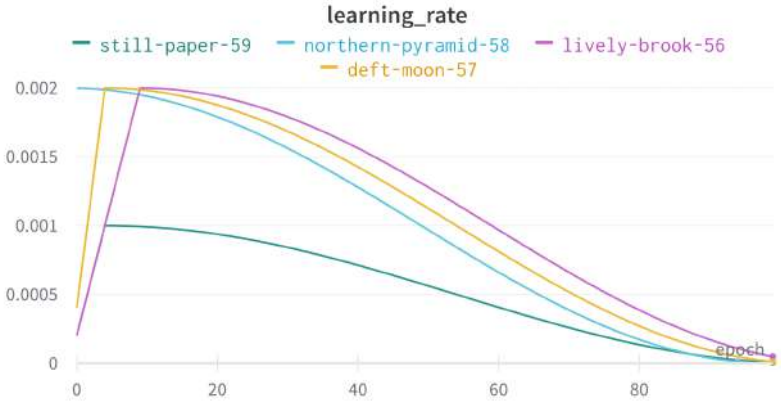


Figure 2.7: Learning rate displayed for various training runs on the W&B dashboard. The schedule is cosine annealing, with varying numbers of warm-up epochs.

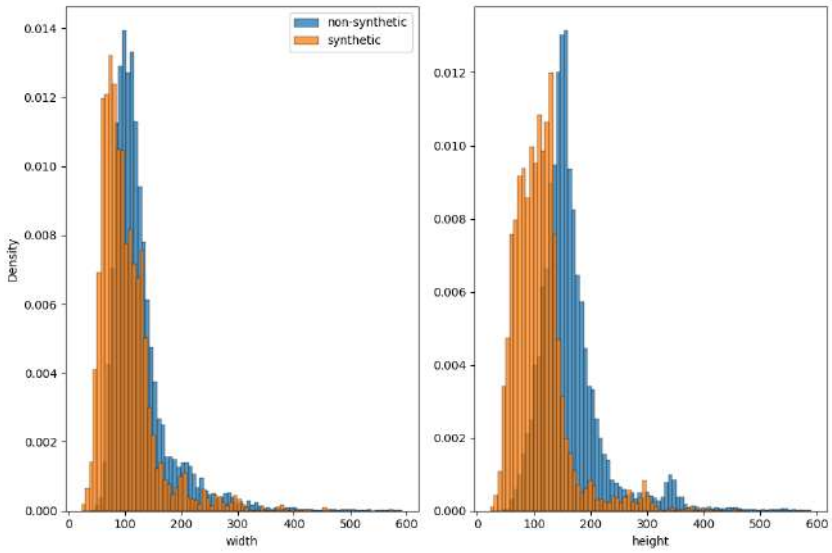


Figure 2.8: Bounding box size distribution for the natural and synthetic datasets.

Faster R-CNN and Retina-Net

To test the newly created synthetic dataset, the same architecture of the model to beat was employed. The new Faster R-CNN with ResNet-50-FPN did great, both in terms of mAP on the (synthetic) evaluation set, and, upon visual inspection, on natural images. Mimicking the training recipe indicated in torchvision, pre-trained weights for the backbone and a multi-step learning rate schedule were adopted, with no augmentations other than the horizontal flip. As the distribution of bounding box sizes displayed in Fig.2.8 is relatively narrow, the number of anchors —a parameter that is not readily accessible— was reduced. This was expected to improve the model speed without affecting its accuracy, but in fact it affected neither of those. In hindsight, perhaps the reason is that, with such high resolution images (4096x2160 pixels),

most of the inference time is taken to compute the feature map. While the performance goals had been achieved, the model was as slow as ever.

Experimentation with RetinaNet-v2 [28], a much more modern architecture compared to Faster R-CNN, was disappointing as there were no improvements neither in speed nor in accuracy. There was good reason to believe that the single-stage model would be faster, but it didn't prove so. Again, this could be due to the high resolution of the images, which make the backbone the main bottleneck. The accuracy was measured on the synthetic evaluation set. However, the assumption that small differences in accuracy on synthetic images proportionally translate to natural images is yet to be verified. Further efforts to improve the accuracy of the model should be delayed until proper test sets are available.

Finally, the Faster R-CNN with MobileNetV3-Large-FPN backbone was tested, with great success. On CPU, the model was **6 times faster** than its equivalent with heavier backbone. Again, using the performance on synthetic data as a proxy for the performance on natural data, hyperparameters such as the number of trainable backbone layers and the learning rate were tweaked to optimal values. An attempt to match this model accuracy with a custom RetinaNet-v2 with MobileNetV3-Large-FPN backbone —a pairing that is not available off-the-shelf— proved unsuccessful.

2.4. RESULTS

The goals of this work were to improve the object detector both in terms of speed and accuracy. While also creating a good basis for further development, these goals were achieved with the creation (and deployment) of two Faster R-CNN models: a slow model (with ResNet-50-FPN backbone) and a fast model (with MobileNetV3-Large-FPN). In the following pages, a visual comparison between the performance of the old and new models is presented. To ensure fairness, a representative subsample of the differences viewed during image inspection is shown. A constant threshold was kept for each model. Each figure displays three images: the first corresponds to the old model, the second to the new slow model, and the third to the new fast model. When only two images are presented, the second image counts for both new models (predictions were identical).

Figure 2.9

Figure 2.10

Figure 2.11

Figure 2.12

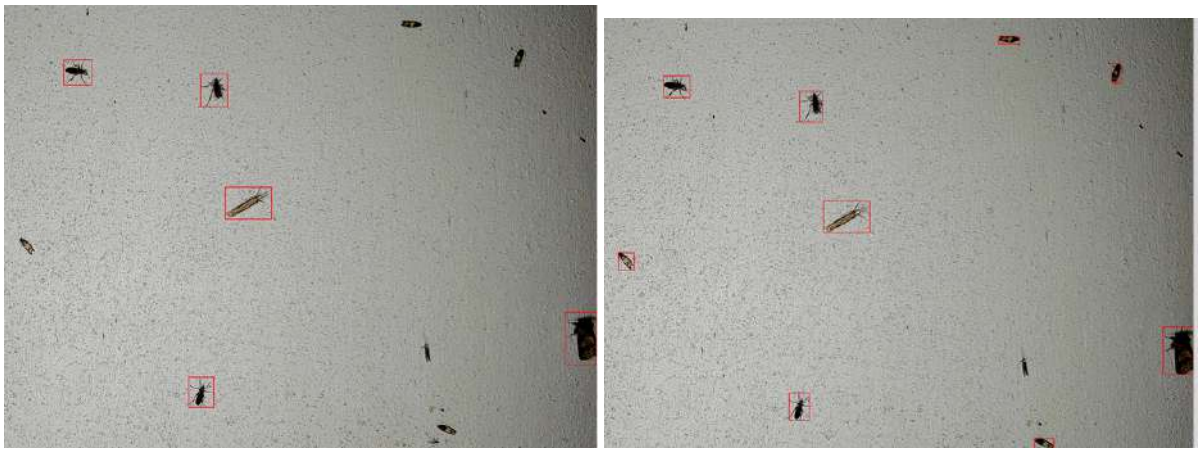


Figure 2.13

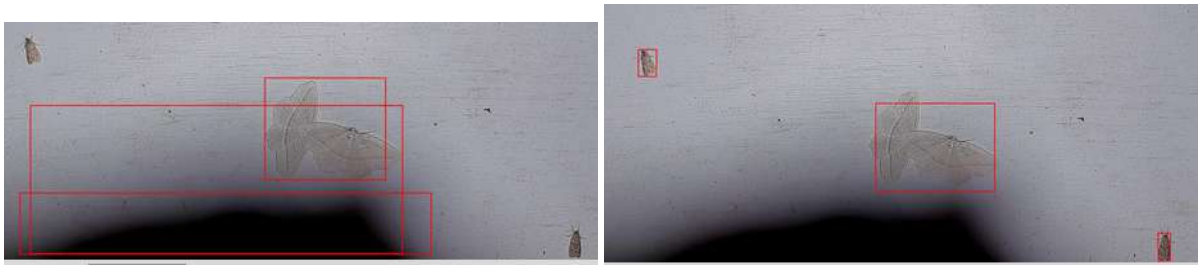


Figure 2.14

Figure 2.15

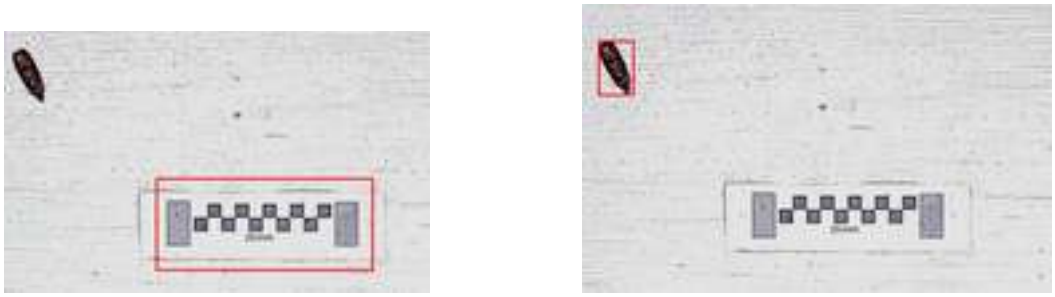


Figure 2.16

Figure 2.17

Figure 2.18

From visual inspection of the models' predictions on natural images, it seems safe to say that both are better than the previous model in terms of accuracy, largely thanks to the new training dataset. The following observations can be made about the new models:

(i) they are better at detecting small moths (Fig.2.13, 2.14, 2.15, 2.16, 2.17, 2.18);

(ii) they are better at separating overlapping moths, although not perfect (Fig.2.9, 2.12);

(iii) they don't get fooled by artifacts on the screen (Fig.2.11, 2.15, 2.16, 2.18;

(iv) they seem more resistant to noisy images (e.g. moths flying close to the camera, Fig.2.10, 2.14);

(v) by design, they tend not to detect mosquitos and flies, as those were present on the background images used for the synthetic dataset, and were filtered out of the segmented crops (Fig.2.18);

(vi) by design, they predict tight bounding boxes;

(vii) the slow model seems to have an edge over the fast mode (Fig.2.9 and 2.11, which is why it was also released. Users can select the desired model based on their time constraints.

The code for the object detection module is available on the ami-ml public GitHub repository. Hopefully, the code for the other modules will soon be merged on the repo as well.

Chapter 3

Active Learning

The final step of the pipeline presented in Fig.1.2 is the moth species classification. As mentioned in the introduction, since there is no diverse and extensive labeled image dataset from moth traps, and given that making one from scratch is not feasible with the current time and budget constraints, the training dataset was constructed with GBIF. For a single region, there are usually multiple thousands relevant species of moths; GBIF has been fundamental to build a classifier able to identify as many classes, making it possible to create training datasets of many hundred thousand images. Unfortunately, there is a significant data shift between training images and images from moth traps.

Typical images from GBIF and from moth traps (following object detection) are presented in Fig.3.1 and Fig.3.2, respectively. In the trap images, the background is uniform, the camera angle is fixed, and the moths fill the frame. Additionally, these images are low resolution (around 120 pixels per side, on average). In contrast, GBIF images are mostly high resolution. The background is heterogeneous, with moths sometimes blending into it; moths are pictured from varying perspectives and often only occupy a small portion of the image. While strong data augmentation operations have been helpful in mitigating the data shift, there is still a drop in species classification accuracy of about 10% from GBIF test images to trap images (a very small set of such images was labeled by experts). One way to tackle the problem would be to incorporate trap images to the training set. Even a relatively small number of these images (e.g. 5% of the training set) is expected to significantly reduce the accuracy drop, and with more and more entomologists becoming interested in the project, the necessary work force to label the images could become available. In this context, active learning appears as a very attractive field of research.

Active learning (AL) encompasses all the techniques that aim to maximize the performance improvement of a model upon labeling of new samples and consequent addition to the training dataset, by selecting the most valuable samples. Hence, it is about making the most of the human annotators' work. The typical workflow is depicted in algorithm 1. In the past decade, with the emergence of data-hungry deep learning, AL has attracted renewed interest, and plenty of techniques have been developed. There are two main categories: uncertainty-based techniques and diversity-based techniques. The previous try to find images the model isn't certain about, as a proxy of the model being wrong about the image. The latter try to find images that best represent the diversity existing in the pool of unlabeled images (e.g. [29]). Finally, there are also techniques that combine both approaches. For an initial exploration of the topic under strict time constraints, precedence was given to the more easily implementable and scalable uncertainty-based techniques.

(a) Eudeilinia herminiata

(b) Acronicta insita

(c) Thysania zenobia

(d) Syngrapha rectangula

(e) Cosmia calami

Figure 3.1: Images from GBIF

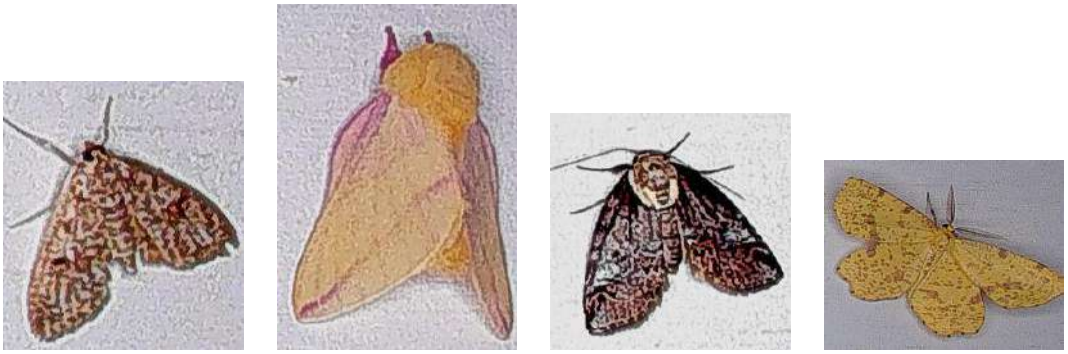


Figure 3.2: Images from moth traps, following object detection

Algorithm 1 The pool-based active learning workflow, which is most common in deep learning. From [30]

Input : Initial training dataset L, unlabeled data pool U, annotators A
Output: A well-trained model M with least labeling cost

1: **while** End condition isn't met **do**
2: Train the model M with L
3: Obtain the representation R of all samples $x \in U, R = M(x)$
4: Query the top-K informative samples K via selection strategies, based on R
5: Annotate the samples K and obtain the labels $Y_K = A(K)$
6: Update $L = L \cup \{K, Y_K\}$, update $U = U/K$
7: **end while**

3.1. MEASURES OF UNCERTAINTY

Neural networks predictions are notoriously unreliable when the input sample is out of the training distribution or corrupted by noise. To effectively apply active learning techniques in deep learning frameworks, better estimations of uncertainty are needed. *Predictive uncertainty*, i.e. the uncertainty related to the prediction $\hat{y}(x^*)$ for a concrete query instance x^*, can be decomposed into two distinct types of uncertainty: *aleatoric uncertainty* and *epistemic uncertainty*. Generally, the previous refers to the notion of randomness, that is, the variability in the outcome of an experiment which is due to inherently random effects. In the context of image classification, aleatoric uncertainty relates to the inherent difficulty of classifying an image (e.g. if the image is blurred or occluded). Epistemic uncertainty refers to uncertainty caused by a lack of knowledge, i.e., to the epistemic state of the agent (e.g. a deep learning model). As opposed to aleatoric uncertainty, epistemic uncertainty can in principle be reduced on the basis of additional information. This should precisely be the goal of uncertainty-based active learning techniques [31]. In other words, the goal is to find images that the model should be able to classify, but can't.

A classic deep learning model typically provides an estimate of aleatoric uncertainty (which, as stated above, can be unreliable). Individually, such a model can't be used to estimate epistemic uncertainty. Instead, it can be done if multiple models are available, by looking at model disagreement. Groups of models are called ensembles, and they are commonly obtained by running multiple trainings: each training is a random process that results in (slightly) different model parameters. Intuitively, if two models disagree on a sample, at least one of them must be wrong, while the other might be right, suggesting that it is possible to classify the sample. Another way to look at it is that models are more likely to disagree on samples that are out-of-distribution, because in these "regions" the decision boundary and the feature space itself are more likely to evolve differently at training. A common estimator for epistemic uncertainty in the context of ensembles is the Jensen-Shannon Divergence, most frequently called Mutual Information. Given the vectors of predicted probabilities $\mathbf{p}^{(e)}$ over the set of classes K for each model e in the ensemble E, the average probability vector $\mathbf{p}$ is computed. The entropy of the average probability vector can be used as a measure of predictive uncertainty:

$$H(\mathbf{p}) = \mathbf{p}^\top \log(\mathbf{p}) \tag{3.1}$$

It can be shown that the entropy is upper bounder by the cardinality of the random variable,

i.e. $H(\mathbf{p}) \leq \log(|K|)$. The mutual information is defined as:

$$J(\mathbf{p}) = H(\mathbf{p}) - \frac{1}{E}\sum_{e\in E} H(\mathbf{p}^{(e)}) \tag{3.2}$$

Since entropy is always positive, the maximum possible value for $J(\mathbf{p})$ is $H(\mathbf{p})$. However, when the models make similar predictions, $\frac{1}{E}\sum_{e\in E} H(\mathbf{p}^{(e)}) \to H(\mathbf{p})$, thus $J(\mathbf{p}) \to 0$, which is its minimum value. This shows that mutual information is higher for samples with high disagreement. An alternative way to look at the formula is that from the predictive uncertainty, we subtract away the expected aleatoric uncertainty, leaving a measure of the epistemic uncertainty.

In the context of active learning, entropy and mutual information are popular scoring functions, also called acquisition functions. These functions are used to evaluate the unlabeled pool of samples. The highest the score, the highest is the expected value of the sample. Other common scoring methods are least confidence, margin sampling and variation ratios. Least confidence is to select the sample with the smallest probability of the top1 predicted class. Margin sampling is to calculate the difference between the probabilities of the top-1 and the top-2 predicted class. These two scoring functions are conceptually similar to the entropy. In contrast, similar to mutual entropy, variation ratios also looks for disagreement between the models, and it is defined as the fraction of members in the ensemble that do not agree with the majority vote. It is worth noting that this function has the undesirable property of only returning a finite number of values, related to the number of models in the ensemble $|E|$ and the number of classes $|K|$. All the implemented scoring functions are summarized in Table 3.1.

Table 3.1: Summary of typical scoring functions in uncertainty-based active learning

Least confidence	$LC(\mathbf{p}) = top1_{k\in\ K}(1 - \mathbf{p}_k)$
Margin sampling	$MS(\mathbf{p}) = 1 - (top1_{k\in K}(\mathbf{p}_k) - top2_{i\in K}(\mathbf{p}_k))$
Entropy	$H(\mathbf{p}) = \mathbf{p}^\top \log(\mathbf{p})$
Mutual information	$J(\mathbf{p}) = H(\mathbf{p}) - \frac{1}{E}\sum_{e\in E} H(\mathbf{p}^{(e)})$
Variation ratios	$V(\mathbf{p}) = 1 - \frac{1}{E}\sum_{e\in E} (\arg\max_{k\in K} \mathbf{p}_k^{(e)} = M)$ where $M = \text{mode}_{e\in E}(\arg\max_{k\in K} \mathbf{p}_k^{(e)})$

3.2. ENSEMBLE CONFIGURATIONS

As mentioned above, ensembles are commonly obtained by running multiple trainings (with different random seeds), usually in the 5-10 range [32] (although theoretically, the larger the ensemble, the better the uncertainty estimation). As the additional computational cost can be a limitation, many techniques have been proposed to derive ensembles without additional training runs. A well known example is Monte-Carlo drop-out [33], where drop-out is applied at test-time and multiple inferences are run for each sample. However, it has been shown that

MC-dropout suffers from mode collapse, i.e. it can lead to a very imbalanced dataset by favoring a specific class during the active learning process [34, 35]. In contrast, in the reported studies, ensembles are able to counteract the imbalance. An interesting approach to derive ensembles with no extra cost is given in [36]. In this work, the disagreement between checkpoints stored during successive training epochs (due to the catastrophic forgetting property of DNNs) is exploited to efficiently construct large and diverse ensembles. While the high computational cost associated to typical ensembles doesn't seem to be an issue at MILA, the encouraging results obtained in [36] motivate exploring different configurations. The configurations are summarized in Table 3.2.

Table 3.2: Summary of the ensemble configurations considered in this study

Name	**Description**
single	A single model is used.
5best	An ensemble of five models, each from a different training run. The best checkpoint is used for each run.
20last	An ensemble of twenty models, obtained at the 20 last epochs of a training run.
5ckpt	An ensemble of five models, obtained from epochs $N, N-5, \dots N-20$ of a single training run, where N is the last epoch. This is a subset of *20last*.

3.3. METHODS

The standard procedure to evaluate active learning techniques follows the typical AL workflow, as described in Algorithm 1, with the difference that labels are usually already available for all the data used in the experiment. Initially, the whole dataset is split into three: an initial training dataset L, a large pool of data U, and a left-out dataset T for testing. The model M (or the ensemble E) is trained on L, evaluated on T, and used to score the pool of data U with the desired function. A number of samples is selected from U accordingly, and added to L. M (or E) is retrained on the updated L, evaluated on T, and so on. The performance measured at each iteration is usually plotted as a function of the number of training samples. Hence, the result of the experiment is a plot with several curves, each corresponding to one acquisition function. Random selection is used as a baseline. With effective active learning techniques, the model is able to closely approach or even reach the performance of a model trained on the full dataset, with only a fraction of the data. The size of the initial training dataset and the number of samples added at each iteration are the main hyperparameters of the experiment.

While the initial intended use for active learning was to select images from the moth traps, it became evident that ongoing efforts to label the images would not deliver enough data within the time at my disposal. However, it was still deemed valuable to develop the active learning framework and to test it on GBIF images. In fact, as the standard evaluation procedure suggests, besides effectively augmenting training datasets, active learning can also be used to effectively reduce training datasets: this operation is called "training data subset search". In [36], using only half of the data, an interesting trick is used to achieve better performance than

the model trained on the full dataset, which is highly-imbalanced: at each active learning iteration, scores are also computed for samples that are already in the training dataset; these can eventually be selected again. At the last iteration, the vast majority of the selected subset is samples with multiple copies. The number of unique images is less than half of those in the full training dataset, yet the model trained on the selected subset outperforms the model trained on the full dataset. It is noted that the effectiveness of the approach is due to its ability to counter the class imbalance. While this experiment was performed in the context of object detection, this approach seems very relevant for our classification problem, as we also face severe class imbalance. The exploration of this method is left as a future research direction.

The dataset used for the experiment consists of around 600k GBIF images gathered for the moth species classifier deployed in Quebec and Vermont (the two neighboring states share the same list of 3150 relevant species). 20% is allocated for the initial training dataset L, 20% is left-out for testing, and the remaining 60% constitutes the pool of data U from which to select new samples. Due to the time constraints, only the first iteration of a typical active learning evaluation is performed. The ensemble configurations listed in Table 3.2 are created by launching five trainings on L with different random seeds, and saving the relevant checkpoints. The model is a standard ResNet-50, and the training recipe (learning rate schedule, weight decay, batch size, data augmentations...) is the same as that of the deployed model, only the number of epochs being adjusted given the reduction in the training dataset size. A python script was developed to score GBIF images with any given scoring function listed in Table 3.1, given an arbitrary list of ResNet-50 models. With that, the score of every image in U is obtained using each ensemble configuration and scoring function.

3.4. RESULTS

3.4.1. Scores distributions

The analysis of the scores gives several insights, and it is also an opportunity to perform sanity checks. First, we can look at scores distributions. The mutual information distributions from *20last* and *5best* is given in Fig.3.3a. As expected, both are long-tailed distributions. This is more pronounced for *5best*, suggesting that this ensemble is more diverse. The distribution for *5ckpt*, not shown for clarity, is slightly closer to zero than *20last*. Given that mutual information's maximal value corresponds to the entropy's maximal value, which is $\log_2(3150) = 11.6$, one can wonder if even the *5best* ensemble is diverse enough. Perhaps a different split between training and validation datasets across the five training runs would have been beneficial.

The entropy distributions from *single* and *5best* is displayed in Fig.3.3b. The two distributions are very similar, except that the distribution from *single* has a 40% higher peak near zero, meaning that the single model has given much more predictions with very high confidence. Unless these predictions are correct, this seems to confirm the idea that a single model's estimation of uncertainty is unreliable.

The margin sampling distribution from *20last* is presented in Fig.3.3c, as a good representation of the distributions from all other ensemble configurations. Surprisingly, there are two peaks: one at the minimum and one at the maximum. This suggests that for many images, it is hard to distinguish between two classes. Finally, the variation ratios distribution is shown in Fig.3.3d, for illustrative purposes. The least confidence distribution (which is also long-tailed, although less so than the entropy's) is omitted for brevity.

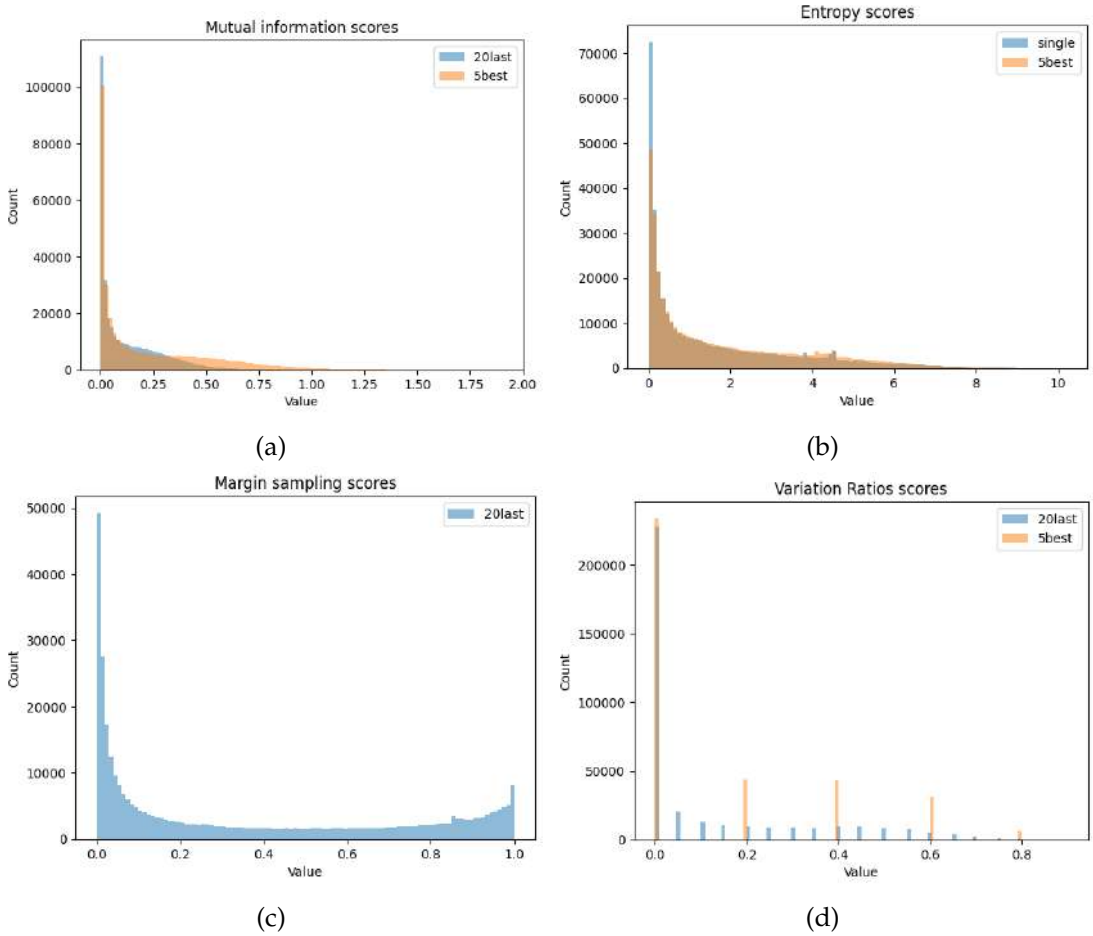
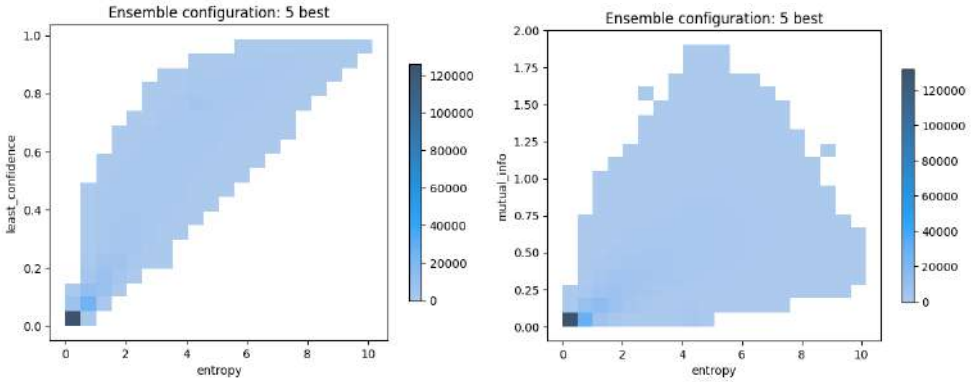


Figure 3.3: Distribution of different scores with various ensemble configurations.

3.4.2. Correlation between scores

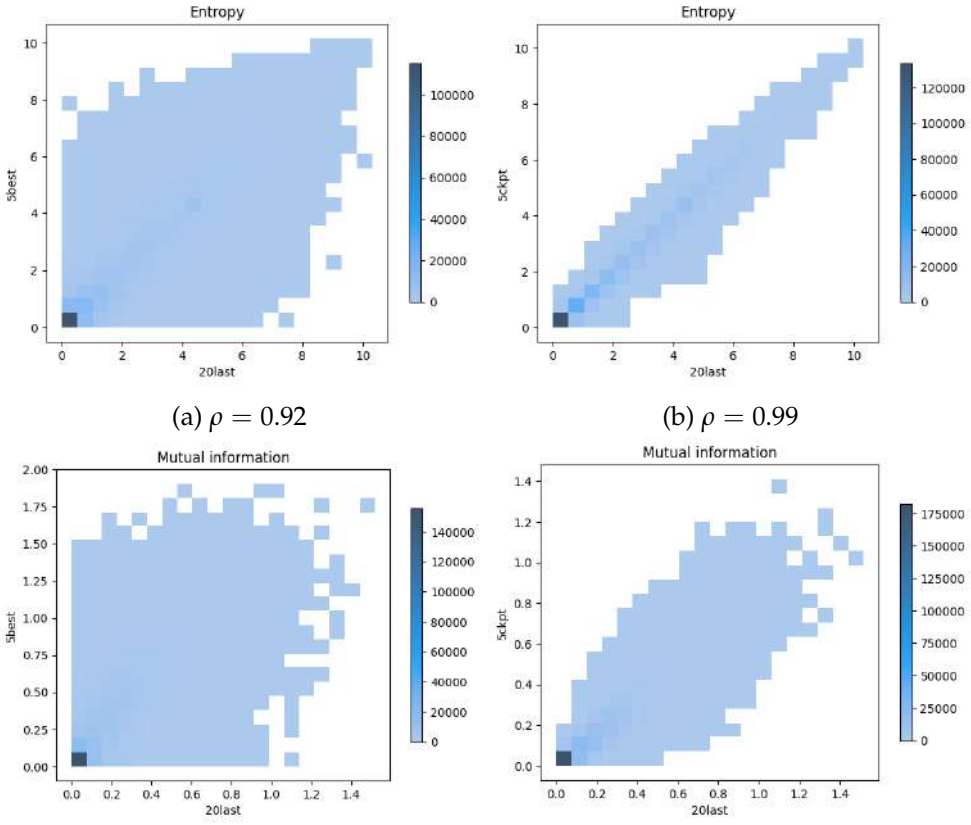
Another point of interest is the correlation between scores. To some extent, this can be displayed in 2D histograms. In Fig.3.4, joint distributions of different scores from the *5best* ensemble configuration are displayed. While the vast majority of samples is concentrated at low scores, for the higher scores least confidence and entropy appear significantly more correlated than entropy and mutual information; this is unsurprising, given that least confidence and entropy are conceptually similar, while entropy and mutual information are fundamentally different. The other two ensemble configurations, *20last* and *5ckpt*, give very similar plots. In Fig.3.5, joint distributions of the same score from different ensemble configurations are presented. As expected, *20last* and *5ckpt* present high correlations, both for entropy and mutual information. In contrast, *20last* and *5best* present lower correlation, especially for mutual information, suggesting that *20last* might not be as good as *5best* (which is the standard ensemble configuration).

Finally, for each scoring function and ensemble configuration, the images in U are ranked, and the top-K are selected to be added to L. K is chosen such that $K = |U|/9 = |L|/3$. For simplicity, only the entropy and mutual information scoring functions are discussed for the rest of the experiment. As a sanity check, the overlap between the newly selected datasets is presented in Table3.3 in terms of Intersection over Union (IoU).



(a) $\rho = 0.99$ (b) $\rho = 0.94$

Figure 3.4: 2D histograms displaying the correlation between different scores, from the same ensemble configuration. Spearman's rank correlation coefficient ρ is given.



(a) $\rho = 0.92$ (b) $\rho = 0.99$

(c) $\rho = 0.87$ (d) $\rho = 0.96$

Figure 3.5: 2D histograms displaying the correlation between the same score, from different ensemble configurations. Spearman's rank correlation coefficient ρ is given.

Table 3.3: Intersection over Union (IoU) between sets selected with different ensemble configurations and three acquisition functions: entropy H, mutual information J, and random

		5best		*20last*		*5ckpt*		*single*	/
		H	J	H	J	H	J	H	random
5best	H	1	0.37	0.56	0.29	0.53	0.27	0.49	0.06
	J		1	0.30	0.38	0.39	0.34	0.26	0.06
20last	H			1	0.32	0.83	0.29	0.66	0.06
	J				1	0.29	0.57	0.26	0.06
5ckpt	H					1	0.27	0.7	0.06
	J						1	0.24	0.06
single	H							1	0.06
/	random								1

3.4.3. Visualization of ranked images

The visualization of top and bottom images in the score rankings is another insightful point. In Fig.3.6, the bottom five images in U according to the entropy and mutual information rankings for the *5best* ensemble are displayed. Three of these images are shared between subsets. Interestingly, the model seems to easily classify images where the same species of moths is presented multiple times. In Fig. 3.8, the top five images in the entropy ranking are shown. These images raise an alarm, as they have nothing to do in a training set for a classifier to be used on trap images such as those in Fig.3.2. In fact, they either don't display a live moth at all, or the moth occupies a minimal portion of the image, on top of being quite blended in the background. The previous are impossible to classify, in other words, they have a high aleatoric uncertainty. Hence, it makes sense that the entropy score is high (and, by the way, the least confidence and margin sampling scores as well), while the mutual information score is relatively low. Finally, the top five images in the mutual information ranking are presented in Fig.3.7. Here, with one exception (which has the highest entropy), the moths are well distinguishable from the background, but again they only occupy a very small portion of the image, which is challenging for the models. Again, these images don't seem helpful at all for a model to be used on the images in Fig.3.2.

In light of these findings, when applied to the GBIF dataset, it is more likely that the uncertainty-based active learning techniques might serve a different purpose than expected: to filter out images that are too difficult and different from the trap images. Tests to see if the selected images would indeed hinder the model's learning —by completing the first iteration of the active learning evaluation— could not be completed in time, and are left for future research.

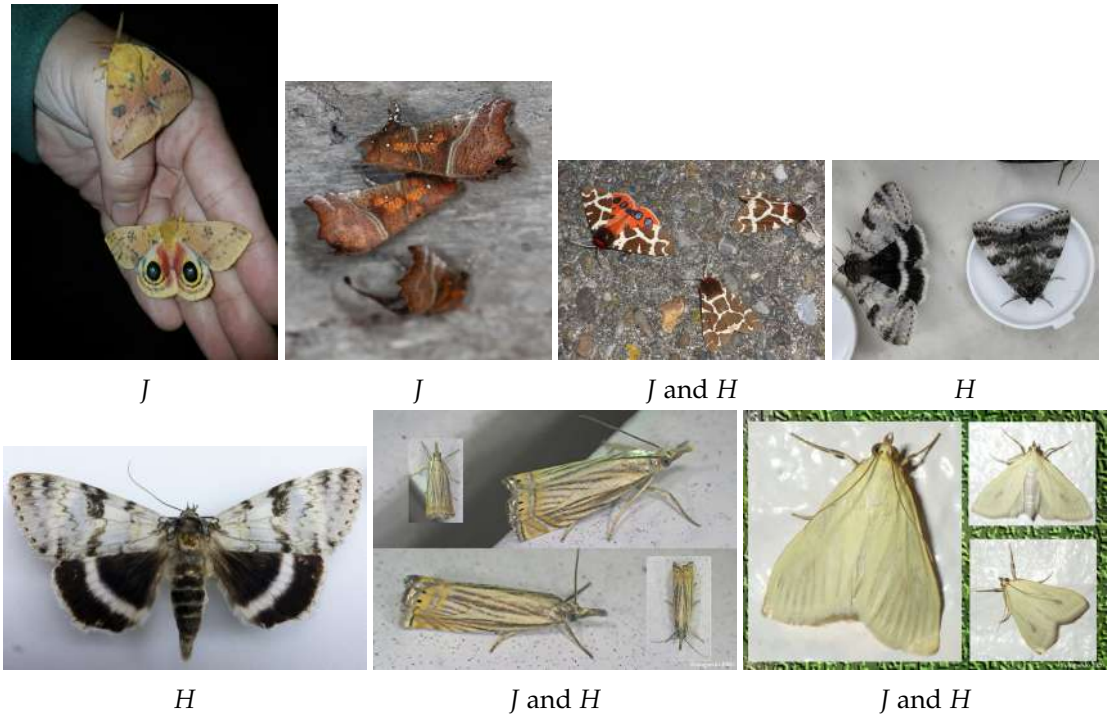


Figure 3.6: Bottom five images in the entropy (H) and mutual information (J) rankings, using the *5best* ensemble. Some images appear in both bottom fives.

Figure 3.7: Top five images in the mutual information ranking with *5best*. The entropy score for each image is given.

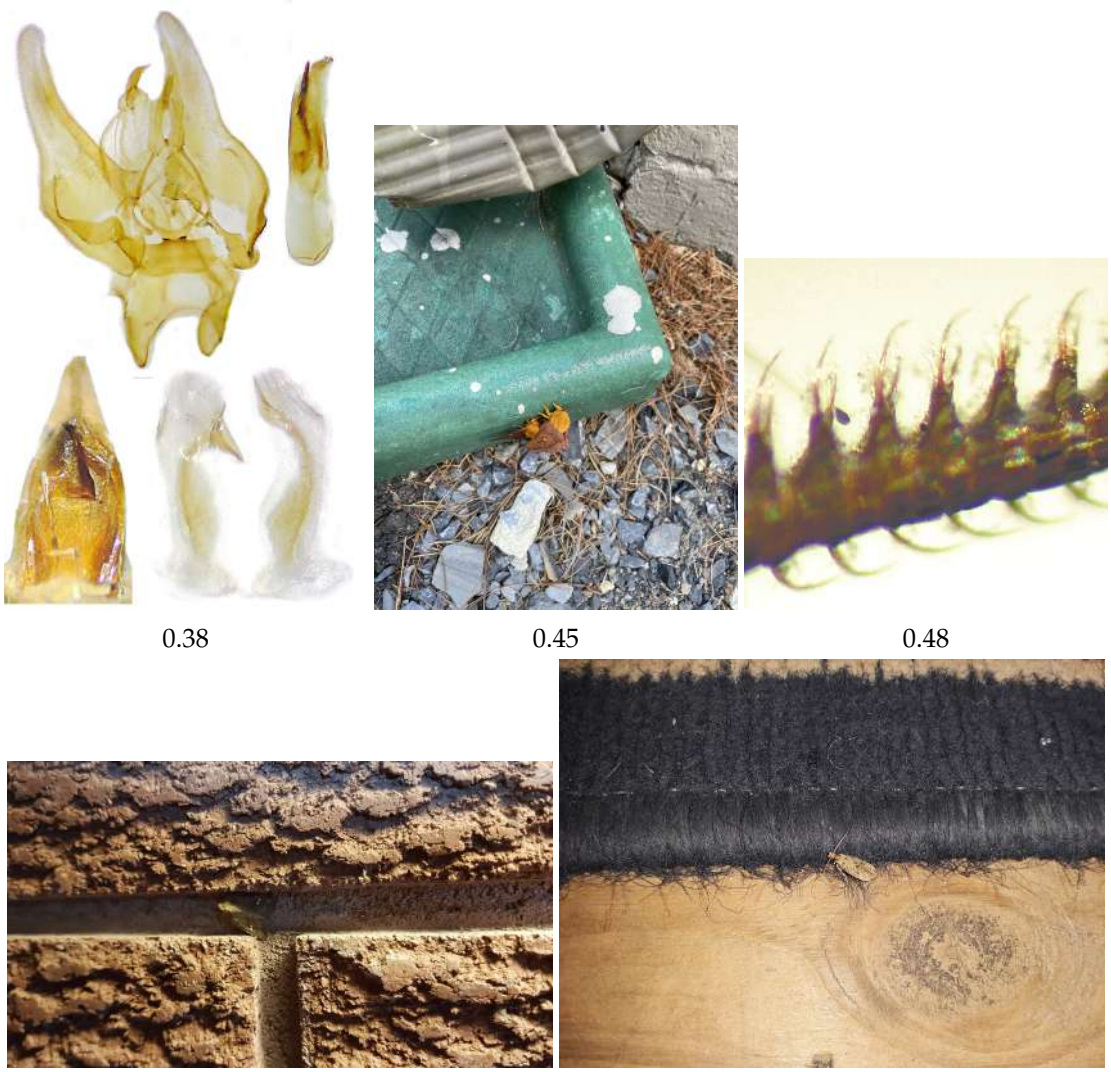


Figure 3.8: Top five images in the entropy ranking with *5best*. The mutual information score of each image is given.

Chapter 4

Conclusion

The main contributions from the work presented in this report are two-fold. First, the object detection module has been significantly improved, both in terms of speed and accuracy. Additionally, good foundations have been laid to facilitate further development of the project. Second, uncertainty-based active learning techniques for the moth species classifier have been explored. While this have been insightful, further research is needed to integrate active learning into the pipeline. In conclusion, it is worth mentioning the two main factors that have negatively affected progress. The first was the lack of testing data. While tricks can sometimes be applied to efficiently create training data —as was done in this project—, there is no getting around the need for manual labor to build testing datasets. In order to reach a significant scale, a coordinated effort is needed. Fortunately, such an effort is currently under way from collaborators both at Mila and outside, and testing datasets will become available in the following months. The second factor has been the lack of effective collaboration inside the team. Considering that the goal —to develop a product— is akin to the one of a start-up, and as collaboration is naturally more challenging when colleagues work remotely from different regions of the world, I believe that the team would greatly benefit from the adoption of project development best practices that are standard in the industry.

Bibliography

[1] Richard G. Roberts, Timothy F. Flannery, Linda K. Ayliffe, Hiroyuki Yoshida, Jon M. Olley, Gavin J. Prideaux, Geoff M. Laslett, Alexander Baynes, M. A. Smith, Rhys Jones, and Barton L. Smith. New ages for the last australian megafauna: Continent-wide extinction about 46, 000 years ago. *Science*, 292(5523):1888–1892, June 2001.

[2] Anthony D. Barnosky, Paul L. Koch, Robert S. Feranec, Scott L. Wing, and Alan B. Shabel. Assessing the causes of late pleistocene extinctions on the continents. *Science*, 306(5693):70–75, October 2004.

[3] Luciano Prates and S. Ivan Perez. Late pleistocene south american megafaunal extinctions associated with rise of fishtail points and human population. *Nature Communications*, 12(1), April 2021.

[4] Sander van der Kaars, Gifford H. Miller, Chris S. M. Turney, Ellyn J. Cook, Dirk Nürnberg, Joachim Schönfeld, A. Peter Kershaw, and Scott J. Lehman. Humans rather than climate the primary cause of pleistocene megafaunal extinction in australia. *Nature Communications*, 8(1), January 2017.

[5] Gerardo Ceballos, Paul R. Ehrlich, and Rodolfo Dirzo. Biological annihilation via the ongoing sixth mass extinction signaled by vertebrate population losses and declines. *Proceedings of the National Academy of Sciences*, 114(30):E6089–E6096, 2017.

[6] Robert H. Cowie, Philippe Bouchet, and Benoît Fontaine. The sixth mass extinction: fact, fiction or speculation? *Biological Reviews*, 97(2):640–663, January 2022.

[7] Anthony D. Barnosky, Nicholas Matzke, Susumu Tomiya, Guinevere O. U. Wogan, Brian Swartz, Tiago B. Quental, Charles Marshall, Jenny L. McGuire, Emily L. Lindsey, Kaitlin C. Maguire, Ben Mersey, and Elizabeth A. Ferrer. Has the earth's sixth mass extinction already arrived? *Nature*, 471(7336):51–57, March 2011.

[8] Aelys M. Humphreys, Rafaël Govaerts, Sarah Z. Ficinski, Eimear Nic Lughadha, and Maria S. Vorontsova. Global dataset shows geography and life form predict modern plant extinction and rediscovery. *Nature Ecology &amp Evolution*, 3(7):1043–1047, June 2019.

[9] Hillary S. Young, Douglas J. McCauley, Mauro Galetti, and Rodolfo Dirzo. Patterns, causes, and consequences of anthropocene defaunation. *Annual Review of Ecology, Evolution, and Systematics*, 47:333–358, 2016.

[10] Arthur D. Chapman. Numbers of living species in australia and the world. Technical report, Australian Biodiversity Information Services, 2009.

[11] Nigel E. Stork. How many species of insects and other terrestrial arthropods are there on earth? *Annual Review of Entomology*, 63(1):31–45, January 2018.

[12] Caspar A. Hallmann, Martin Sorg, Eelke Jongejans, Henk Siepel, Nick Hofland, Heinz Schwan, Werner Stenmans, Andreas Müller, Hubert Sumser, Thomas Hörren, Dave Goulson, and Hans de Kroon. More than 75 percent decline over 27 years in total flying insect biomass in protected areas. *PLOS ONE*, 12(10):e0185809, October 2017.

[13] Sebastian Seibold, Martin M. Gossner, Nadja K. Simons, Nico Blüthgen, Jörg Müller, Didem Ambarlı, Christian Ammer, Jürgen Bauhus, Markus Fischer, Jan C. Habel, Karl Eduard Linsenmair, Thomas Nauss, Caterina Penone, Daniel Prati, Peter Schall, Ernst-Detlef Schulze, Juliane Vogt, Stephan Wöllauer, and Wolfgang W. Weisser. Arthropod decline in grasslands and forests is associated with landscape-level drivers. *Nature*, 574(7780):671–674, October 2019.

[14] Roel van Klink, Diana E. Bowler, Konstantin B. Gongalsky, Ann B. Swengel, Alessandro Gentile, and Jonathan M. Chase. Meta-analysis reveals declines in terrestrial but increases in freshwater insect abundances. *Science*, 368(6489):417–420, April 2020.

[15] Graham A. Montgomery, Robert R. Dunn, Richard Fox, Eelke Jongejans, Simon R. Leather, Manu E. Saunders, Chris R. Shortall, Morgan W. Tingley, and David L. Wagner. Is the insect apocalypse upon us? how to find out. *Biological Conservation*, 241:108327, 2020.

[16] G. W. Hopkins and R. P. Freckleton. Declines in the numbers of amateur and professional taxonomists: implications for conservation. *Animal Conservation*, 5(3):245–249, August 2002.

[17] Erica Fleishman and Dennis D. Murphy. A realistic assessment of the indicator potential of butterflies and other charismatic taxonomic groups. *Conservation Biology*, 23(5):1109–1116, October 2009.

[18] Kim Bjerge, Jakob Bonde Nielsen, Martin Videbæk Sepstrup, Flemming Helsing-Nielsen, and Toke Thomas Høye. An automated light trap to monitor moths (lepidoptera) using computer vision-based tracking and deep learning. *Sensors*, 21(2), 2021.

[19] Michael J. Furlong, Denis J. Wright, and Lloyd M. Dosdall. Diamondback moth ecology and management: Problems, progress, and prospects. *Annual Review of Entomology*, 58(1):517–541, January 2013.

[20] Maartje J. Klapwijk, György Csóka, Anikó Hirka, and Christer Björkman. Forest insects and climate change: long-term trends in herbivore damage. *Ecology and Evolution*, 3(12):4183–4196, September 2013.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[23] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[24] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[27] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[28] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z. Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection, 2020.

[29] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach, 2017.

[30] Mingfei Wu, Chen Li, and Zehuan Yao. Deep active learning for computer vision tasks: Methodologies, applications, and challenges. *Applied Sciences*, 12(16), 2022.

[31] Vu-Linh Nguyen, Sébastien Destercke, and Eyke Hüllermeier. Epistemic uncertainty sampling, 2019.

[32] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2017.

[33] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data, 2017.

[34] William H. Beluch, Tim Genewein, Andreas Nurnberger, and Jan M. Kohler. The power of ensembles for active learning in image classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9368–9377, 2018.

[35] Remus Pop and Patric Fulop. Deep ensemble bayesian active learning : Addressing the mode collapse issue in monte carlo dropout via ensembles, 2018.

[36] Kashyap Chitta, Jose M. Alvarez, Elmar Haussmann, and Clement Farabet. Training data subset search with ensemble active learning, 2020.