Saving the Rhino Hermes

Alice Hua
Section 2
alicehua11@berkeley.edu

Kevin Martin
Section 2
kmart757@berkeley.edu

Catherine Mou
Section 2
catherine041616@berkeley.edu

Eugene Shen
Section 3
eugene.shen.y@berkeley.edu

Nicole Chen
Section 3
nchen92@berkeley.edu

08/05/2021

Abstract

This study attempts to build a proof of concept for better rhino protection in the wild. In addition to the perpetual threat of habitat loss, the majority of rhino subspecies are considered critically endangered due to persistent poaching for their horns (used in traditional medicine) or killed for trophies in Africa and Asia. The need for better rhino protection is urgent given that up to 95% of the rhino population has declined since the beginning of the 20th century. Some rhino subspecies such as the North White rhinos are functionally extinct [1,2]. Our study works with WildTrack [3], a non-profit organization, to understand how ground efforts to protect endangered rhinos within the Kuzikus Wildlife Reserve in Namibia could be enhanced with the use of deep learning and edge devices. This proof of concept provides a potential solution with an end goal of deploying a real-time object detection model from the edge while providing in sync notifications on the cloud for autonomous drone fleet monitoring of rhino populations.

Introduction

Our task is to train a YOLOv5 object detection model to identify rhinos from multiple drone altitudes so that the anti-poaching units can effectively be deployed and poachers can be caught on time. We curated a unique dataset composed of primarily labelled high and low altitude (30-70m) WildTrack and web-scraped YouTube drone videos as well as San Diego Zoo still cam clips. Several novel techniques are used to train the models to overcome a fundamental challenge that the training dataset is small and the objects within the set tend to be far away.

Modern day drones have advanced to a state where obstacle avoidance sensors are integrated to their mounted 4K cameras, but have yet to make real-time streaming of autonomous flights over the network reliable and affordable. Ideally, inference would take place on the drone as the model ingests the livestream capture by the drone camera. Alerts for the frames in which animals were detected in, the predicted bounding boxes, and location coordinates are then sent to the cloud to notify the end-user about the animal's whereabouts on a map. Due to the current state of the industry in which our Anafi Parrot drone [4] allows only one private Wi-Fi connection, our alternative proposed pipeline is to use the Jetson Xavier NX [5] as an embedded module for our model deployment on the drone.

Literature Review

Students from MIDS have worked with WildTrack previously to identify animals based on their footprints [6] and attempted segmentation of animal trails. Our project builds on these prior efforts with a focus on animal object detection. We attempt to meet the challenges of small object detection and working with sparse datasets of animals in the wild.

Another project, WildEyes AI, also had a similar goal of wildlife detection by utilizing an object detection model which was embedded on the edge with motion sensor equipped cameras hidden strategically among trees. These cameras remain asleep until a motion is detected, after which they transmit images over cellular networks (or over radio, for areas without network coverage) [7]. The drawbacks of this project, which our drone-based project can help address, is the prohibitive placement cost of multiple motion cameras, the inability to scale wildlife detection across vast regions of wildlife reserves and missing poaching activity occurring within unmonitored areas..

There is another computer vision competition on drone imagery with a large scale dataset called VisDrone. Images from this set are composed of urban and suburban areas of different cities in China. The challenge promotes establishing benchmarks in four tracks: image and object detection, single and multi-object tracking [8]. However, we were unable to leverage this dataset because our model is trained specifically on detecting animals and humans in the wild.

Challenge

We face the challenges of small object detection on images with varying resolution sizes, and the lack of a readily available large training dataset. Objects of interest often appear small on drone imagery, which can be taken up to 120m from ground level. There is a paucity of drone related studies, with limited availability of open-source datasets. Of those found, most focused on identifying traffic and humans within urban environments [8,9]. This meant that we could not easily externally augment our training dataset. Hence, we looked at other possible methods to enhance the accuracy of small object detection including: increasing the input resolution to the model, image tiling [10], data augmentation and filtering out extraneous classes [11].

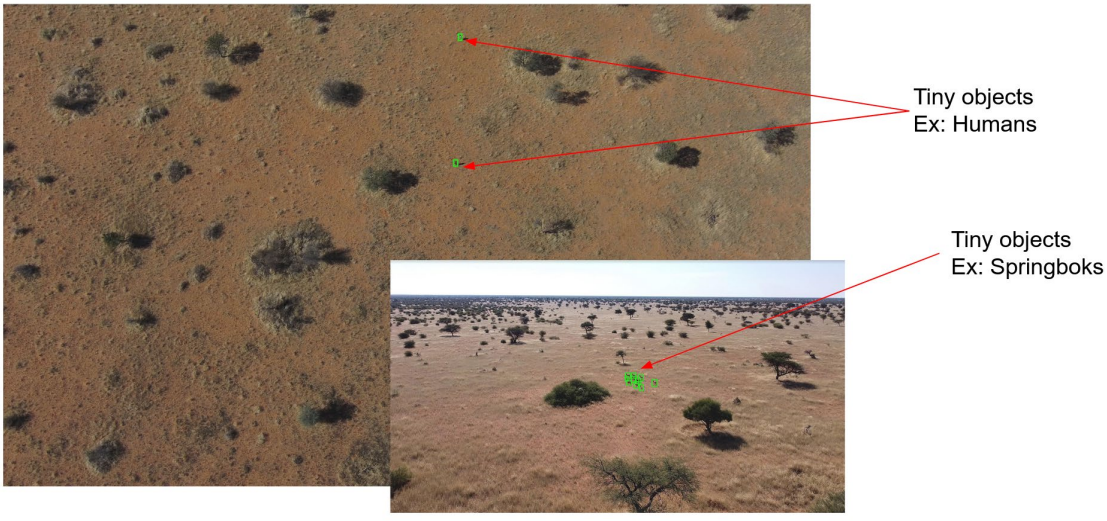

Figure 1: Examples of small object detection challenge

Dataset

The data was collected from several different sources. These included drone and video images obtained from WildTrack, publicly available drone images obtained from the internet, still cam footage from the Giraffe Cam at the San Diego Zoo Safari Park, and drone footage personally collected using our Parrot Anafi drone.

Source	Size	Altitudes
Drone footage directly from WildTrack and Kuzikus	4000x3000 6000x4000 3840x2160 4608x3456	30 - 70 meters
Drone footage extracted from youtube videos	1280x720 1415x588 1920x1080 1518x1113	10 - 20 meters
Drone footage from Parrot Anafi	3840x2160	30 - 70 meters
Still cam footage from SDZ	1280x720	n/a
Offline & Online augmented	501x282	Generated from 30-70m drone images

Figure 2: Data sources along with corresponding frame size and altitude it was collected from

Pre-Processing

We used OpenCV and FFmpeg to extract frames from video and used MakeSense.ai to label bounding boxes for five classes - rhino, giraffe, ostrich, springbok and human. We incorporated a quality control process via a separate notebook which ran through labelled frames and attached the annotated bounding boxes on their corresponding frames. This allowed us to inspect for the four most common mistakes: ghost labels, labels that were too loose, labels that were too tight, and missing labels (see Appendix Figure 9). We show the training-validation set distribution of our five classes (right), and the 80/20 split based on videos to prevent data leakage (left) in Figure. 3 below.

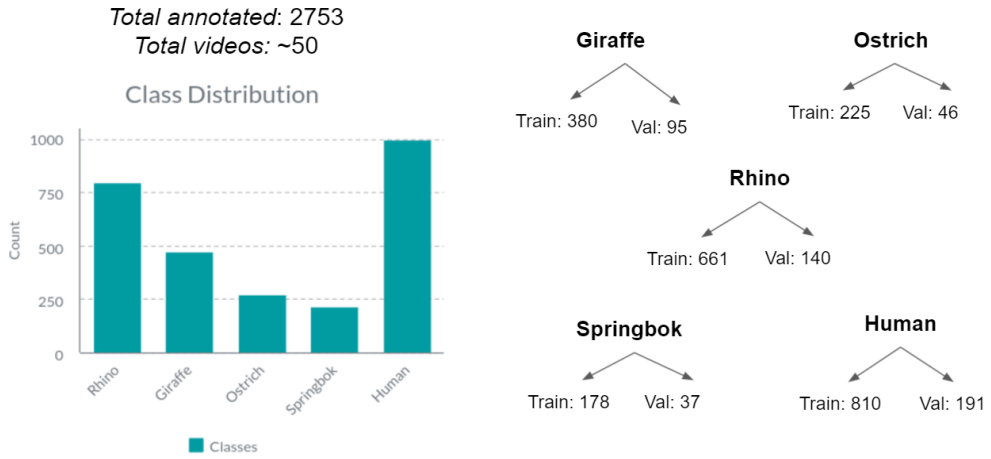

Figure 3: Classes and distribution into train and validation sets

Offline Augmentation

Single Image Generative Adversarial Network (SinGAN) [12] was used due to the limited number of videos and images available, especially with the correct background environment, perspective, and altitude. SinGAN works by capturing the internal distribution of patches within a target image and generating samples that carry the same visual content.

The training time is dependent on the size of the starting image and the maximum size specified for the generated images. Most of the images we were working with were large-scale high-resolution images, so training the SinGAN took a significant amount of time. We trained 17 images to generate a total of 82 rhino images and 44 ostrich images that were used in our train set. For SinGAN training, most images took an average of 8 to 10 hours on a g4dn.4xl AWS instance. We found that SinGAN performed best on images that were at a high enough altitude that there were not too many fine details to train, yet low enough altitude that the objects we were interested in were not lost in the background.

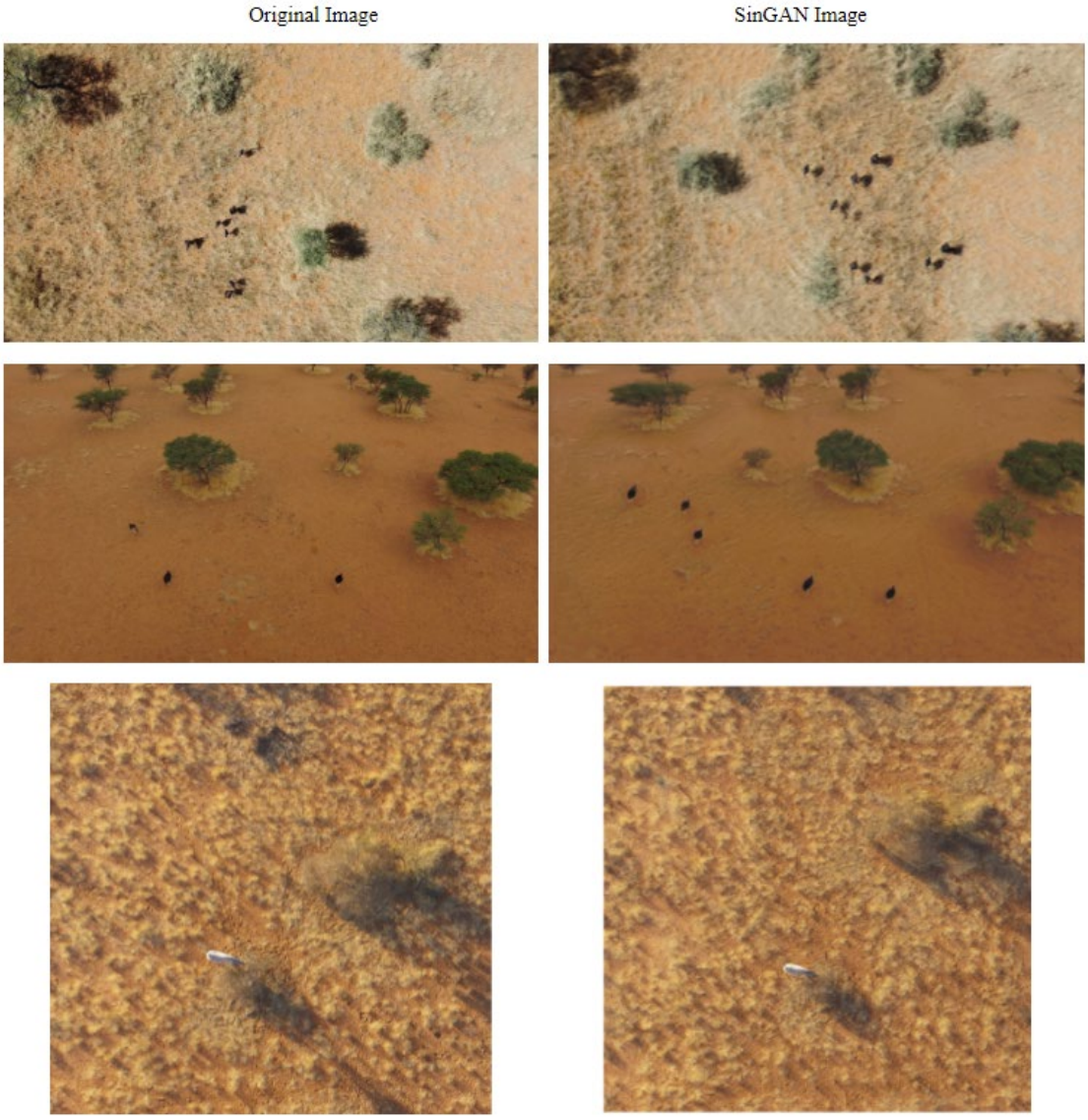

Figure 4: SinGAN examples of ostriches and rhino

Beyond normal SinGAN we also tried the super resolution function available with SinGAN. It works by iteratively up-sampling the train image and feeds it into SinGAN's finest scale generator. However, this technique yielded poor results on rhino images. We suspect that this was due to the rhinos not being in the foreground or the focus of the image, resulting in SinGAN super resolution learning that the small rocks in the foreground were more important and adding more small rocks rather than rhinos to the generated images. Due to the poor results and the huge compute resources required to generate a single output image, we decided not to pursue this approach (see Appendix Figure 10).

In addition to SinGAN, we also utilized Photoshop to manipulate images. In order to reduce false positives, we made sure to include background images with no objects in them [13]. For some videos, we were not able to extract frames with no objects in them. Instead, we photoshopped our target objects out of these images in order to generate background images for training.

Another way we tried to use Photoshop was to generate synthetic images to train on. We photoshopped objects from one image and transferred them to other images. The results can be seen in Figure 11a. These images appear convincing to the human eye, but are more time consuming to create. To try to expedite the process, we tried photoshopping an image with all target objects and running SinGAN on the synthetic image. The results of this can be seen in Figure 11b. In the example generated image, it is possible to determine that there are some objects due to the shadows, but it is extremely difficult to determine the correct label for many of the objects generated. Further fine tuning of the input photoshopped image could potentially generate better SinGAN results, but we had limited time to test this further.

Model

In general, CNN-based object detection methods can be divided into 3 main groups: region based, regression-based, and single shot detectors [14]. Region based detectors like Faster R-CNNs are the most accurate but the slowest while regression based detectors like YOLO have the least accuracy amongst the 3, but have the fastest performance times.

We utilized YOLO as our real-time object detection model of choice for the following reasons:
- Fastest inference speed, which is essential for real-time video feeds
- Reasonable accuracy metrics enough for the model to afford to be more sensitive (better recall), rather than specific. Having a sensitive model is more important than specificity, as the web alerts will be reviewed by the user

We decided to use YOLOv5 rather than the earlier versions of YOLO due to the following reasons:
- Implementation in Pytorch rather than C, with clear Pytorch documentation and ease of use
- Faster training and inference duration & smaller model size
- Flexibility to utilize pre-built optimized YOLOv5 models based on image size [15]
- Ability to half the floating point precision for inference speed, auto learning bounding box anchors [16]

The YOLOv5 model architecture consists of the Backbone, Neck and Head. Cross Stage Partial Network is used as the model Backbone to extract important features from the given input image. PANet is used for the model Neck to generate feature pyramids. Feature pyramids help models to generalize on object scaling and help to identify the same object with different sizes and scales. The model Head is mainly used to perform the final detection by applying anchor boxes on features and generating final output vectors with class probabilities, objectness scores, and bounding boxes [17]. We use the default settings for YOLOv5, with SGD as the optimizer, a predefined set of activation functions and BCEWithLogitsLoss for the loss function.

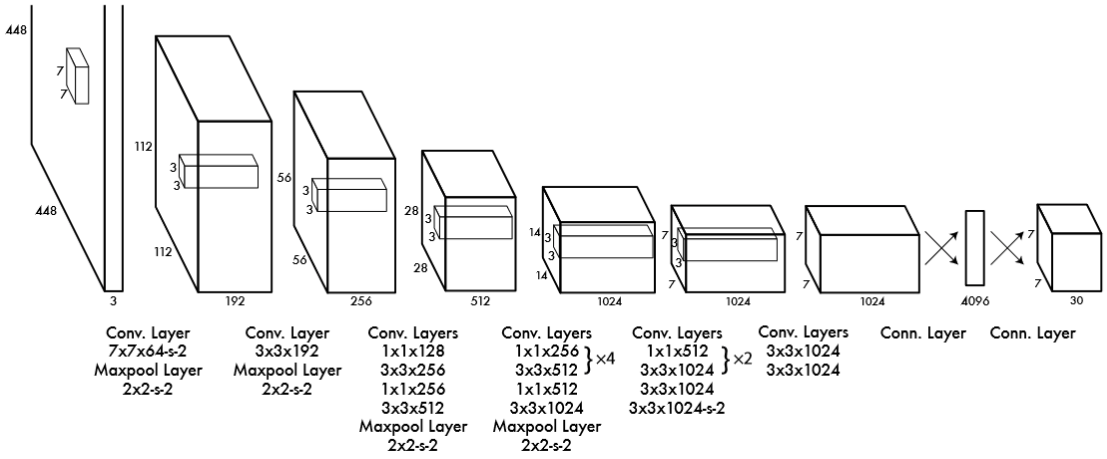

Figure 5: Model architecture of the original YOLO model [18]

Online Augmentation

YOLOv5 also comes with tools that enable online augmentation, via hyperparameter inputs and using the Albumentations package. Methods to augment the dataset, such as copy paste, mixup, mosaic and alterations of the image saturation can be easily performed. We use Albumentations to randomly flip and apply various forms of blur to the input images.

Tiling

The images in our dataset generally suffer from an issue known as the “small object problem”. The objects that we want to identify are very small relative to the size of the image. This is a classical hurdle in object detection algorithms where objects that are small in absolute terms of pixels and relative to the size of the input image have a very difficult time being classified [10]. The reasons behind this are two-fold [11]. First, the input images must be compressed to whatever size the model is set up to take before they can be used. This means that there are less pixels and therefore less information for the algorithm to be able to detect objects.

Secondly, object detection algorithms rely on anchor boxes to make initial guesses of where objects are. If the bounding boxes are very different from the size that the algorithm was originally trained on, it has a hard time making good initial guesses with these anchor boxes. Luckily, the YOLOv5 algorithm that we used has a custom function to learn anchor boxes [19]. So, for our modeling the anchor box problem is not as much of an issue, but the compression problem remains. We address this with a technique called tiling.

Tiling is a process of breaking up a large image into smaller pieces then running those pieces through the model. Bounding boxes are kept and scaled accordingly so that the object of interest remains boxed. Generally, the tiles are overlapped on the image so that objects at the border between tiles are kept intact. This solves the small object problem because the smaller tiled images do not need to be compressed to be used by the model. They maintain their full fidelity and are a larger portion of the image relative to the size of the tile being passed in. This allows greater precision in making predictions. Additionally, the higher quality images are easier and more meaningful for the model to train on.

We implemented tiling during the training phase and did not implement tiling at the inference phase due to time constraints. First we considered that a high quality 720p video frame comes in at 720x1280 in size. With such video quality, small objects would still likely be visible. Hence, we set our YOLO model to take in images at 1280 pixels.

We tiled our images at 1280x1280 with a 320 overlap between adjacent tiles. The ¼ of tile overlap comes from a paper by Unel et al. [10] and helps prevent objects from being lost at the boundary between tiles. This helped break up some of the ultra high-definition images in our dataset (some were 4000 pixels wide) and maintain small objects at a reasonable size (see Appendix figure 12).

Where objects were partially cut off by the tile, we decided to only keep the box for training if more than 50% of the bounding box for the object was contained on our tile. This prevented potentially irrelevant bounding boxes with limited or no info from appearing at the edges of the tiles. The tiling implementation itself was a heavily modified version of a script from GitHub [20].

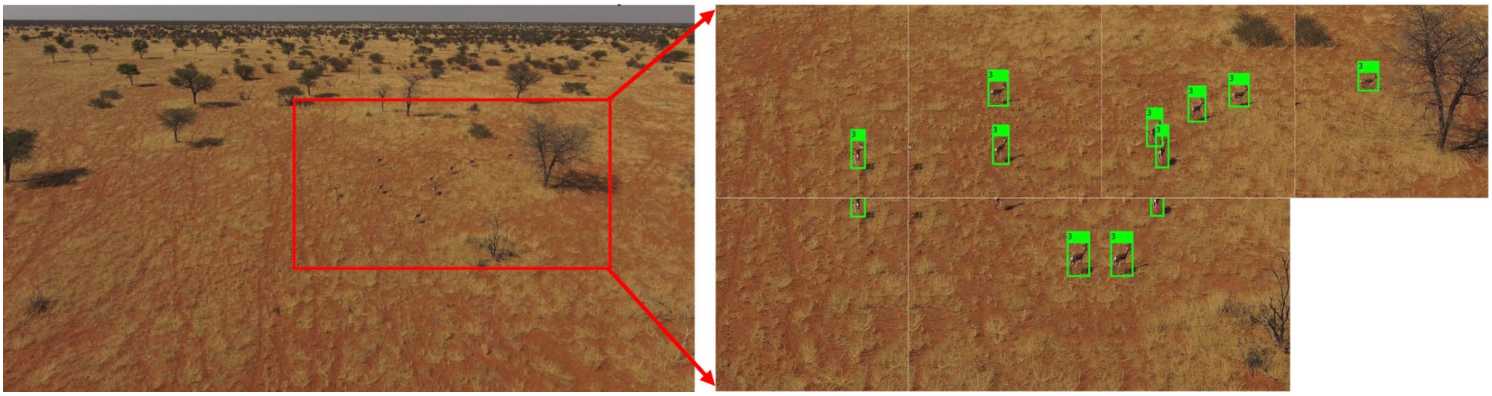

Figure 6: Example of a springbok image (3840x2160) tiled

Training progression

Due to the short window of time accorded to the project, we performed model training iteratively, as the primary dataset was being built from scratch and enhanced with subsequent additional videos of drone footage from WildTrack. We started by training the lightest YOLOv5s model on our initial dataset with an input image size of 992 and assessed the difference between a 3-class (rhino, other and human) vs 5-class dataset. No additional augmentation was performed. We found that a 5-class dataset provided better performance for rhino identification, which was our project's main goal.

We subsequently added additional train images to the dataset and started tuning possible data augmentation and regularization parameters such as color saturation, mixup, copy-paste, and label smoothing. We also experimented with increasing the IoU threshold from the default value of 0.2 to 0.45. Image size was kept at 992. With these series of tuned parameters, we saw a noticeable increase in performance across all 5 classes. We required the use of multi-GPU instances from this point onwards to complete training in a timely manner.

We then updated our model to a larger YOLOv5l6 model which was pre-optimized to ingest images with a larger image size of 1280. We experimented with the larger image size, to minimize the loss in feature space when a large image (e.g. 6000x4000) was scaled down by the model upon being ingested. We found that with the same settings and compute

resources, a pre-optimized model trained faster, compared to forcing YOLO to ingest a larger image size than it was originally not optimized for (YOLOv5s was originally pre-optimized to ingest an image size of 640). We also noticed a slight difference in individual class performance.

The train dataset was subsequently enhanced with the addition of images from new videos and also images generated using GAN. We ran this final dataset on the prior model to establish an intermediate baseline. We then continued to tune the regularization parameters (copy-paste and label smoothing), to achieve improved performance across all 5 classes. Our next phase of training involved using features from the Albumentations package which could be activated via a direct integration with YOLOv5. Numerous features were experimented upon, and our final series of albumentation features utilized were primarily related to changes in image positioning (flipping and rotating) and various forms of blurring (motion blur, median blur and gaussian noise). We saw performance improvements across 4 out of the 5 classes.

Our final model (which also gave the best performance) was run with the above parameters, on a train dataset which was tiled. Tiling was performed on images larger than 1280 in size, with such tiles added to the original dataset. The tiled images allowed the model to preserve the feature space of labelled objects without the degradation that accompanies image downscaling. Validation was performed on the original un-tiled dataset to test its performance upon varying image sizes (including the larger-sized ones). The use of tiling more than doubled the train dataset size and necessitated the use of the largest 96 VCPU 8-GPU P-instance for training.

Interesting take-aways from training:

1. **Size of the training data on accuracy**
   Ostrich and Springbok classes had the lowest mAP which corresponded to the lowest number of available train images by class. We saw how mAP improved with each cycle of addition to the image count for the Rhino, Giraffe and Human classes.

2. **Efficiency in using pre-optimized input image size**
   Although YOLO can take in any input image size in multiples of 32, we realized that there was significant speed boost when pre-optimized input images sizes were used. Using a smaller input image size was slower when used with a model not optimized for it, compared to using a larger input image size on a model pre-optimized for it.

3. **Using black and white images**
   We found a consistent improvement in performance when the HSV saturation parameter was tuned to 0.0, which produced black and white images. This was counter to the natural assumption that having greater feature inputs in the form of color variation would help in model performance. However, we suspect that using black and white could help the model in differentiating the shadows of the animals better. Depending on the time of the day the footage was shot, shadows are predictive of animal classes (see Appendix Figure 13).

4. **Easy access to image augmentation and regularization tuning**
   YOLOv5 contained an easily accessed set of arguments which could be used to easily tune the image augmentation and regularization parameters during training. We hypothesize the use of mosaic, mixup and copy-paste helped the model learn better, as most images contained small objects which occupied a small sub-section of the given image. Label smoothing allowed the model to learn around greater uncertainty, as the differences in how wildlife and humans appear are less obvious with high altitude aerial imagery, as opposed to typical face-level, close-up pictures.

5. **Tiling is useful but expensive**
   Tiling increased model performance by preserving the feature space of objects within larger (>1280 pixels) images. Using tiling to augment our data and not doing the tiling at inference saw mAP improve on the rhino class by 3%. However, there was increased compute resource required to run the model against a significantly larger dataset. Tiling the train dataset did not cost much time as the tiling process is optimized to use multi-threading. On the other hand, if compute resources for training are not limited, tiling might help reduce the impact of having a relatively smaller dataset. Tiling also allows the team to reduce the work associated with annotation and the drawing of bounding boxes around very small objects compared to the use of GANs to augment the train dataset size.

Evaluation Metric

The intersection over union (IoU) computes the intersection over the union of the bounding box for the ground truth and the predicted bounding box. An IoU of 1 indicates that the bounding boxes overlap perfectly. For our model, we adjust the IoU threshold as a hyperparameter, where positive results above the IoU threshold are classified as true positives, and those below the threshold are false positives.

Based on the true positives, false positives and false negatives, we derive the precision and recall for all objects to construct the precision recall curve. We use the commonly applied mean average precision (mAP) for the threshold IoU at 0.5 and from 0.5 to 0.95 as our outcomes of interest. The mAP can be generated from the precision recall curve using the 11 point interpolation technique [21]. The mAP 0.5:.95 corresponds to the average precision value of each IoU with thresholds from 0.5 to 0.95, using a step size of 0.05.

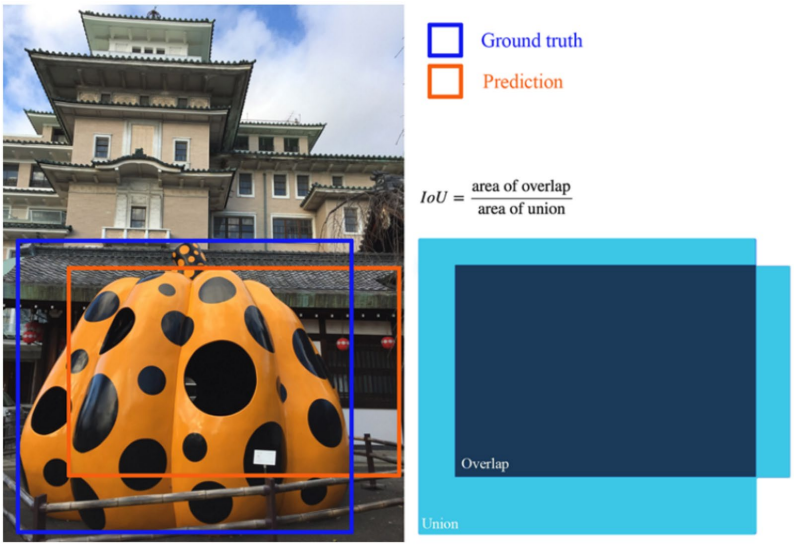

Figure 7: IoU visualized [22]

Results

Class	Model	GAN	Online Augs	Image Size	Tiles	Dataset Size	mAP0.5	mAP 0.5:.95	Training duration
Rhino Other Human	YOLOv5s	None	None hsv_s 0.7 mixup 0.0 copypaste 0.0 iou_t 0.2 lab_smooth NA	992	None	2276	62.7 36.8 23.2	34.0 15.5 10.6	**3 hrs** on 1-GPU
Rhino Giraffe Ostrich Springbok Human	YOLOv5s	None		992	None	2276	67.7 66.7 35.8 7.34 20.2	34.3 30.9 15.7 2.02 8.76	**3 hrs** on 1-GPU
	YOLOv5s	None	hsv_s 0.0 mixup 0.5 copypaste 0.0 iou_t 0.45 lab_smooth 0.5	992	None	2770	78 67.3 45.1 8.65 32.5	43.4 42.7 16.6 1.65 17.7	**9.5 hrs** on 4-GPU Single multi-card
	YOLOv5l6	None		1280	None	2770	77 73.1 35 5.03 38.3	51 44.6 16.9 1.23 19.5	**8 hrs** on 4-GPU Single multi-card
	YOLOv5l6	Yes		1280	None	3019	73.5 69.1 39.8 9.92 37.4	48.6 44.7 15.5 2.46 20.2	**2.45 hrs** on 8-GPU Single multi-card
	YOLOv5l6	Yes	hsv_s 0.0 mixup 0.5 copypaste 0.2 iou_t 0.45 lab_smooth 0.1	1280	None	3019	78.4 69.2 41.0 19.7 39.9	48.2 42.5 15.9 5.05 20.0	**6.7 hrs** on 4-GPU Multi-machine multi-card
	YOLOv5l6	Yes	hsv_s 0.0 mixup 0.5 copypaste 0.2 iou_t 0.45 lab_smooth 0.1 A.HorizontalFlip A.RandomRotate90 A.VerticalFlip A.MotionBlur A.MedianBlur A.GaussNoise	1280	None	3019	78.1 65.8 40.2 22.2 46.3	49.5 38.6 13.5 5.81 21.9	**2.5 hrs** on 8-GPU Multi-machine Multi-card
	YOLOv5l6	Yes		1280	Tiles: 4390 Tile size: 1280	7409	81 82.6 33.6 24.6 41.8	53.5 57 10.3 6.912 21	**4.8 hrs** on 8-GPU Multi-machine Multi-card

Edge

For our test set, we curated a test video made up of a holdout set of WildTrack videos, the San Diego Zoo still cam [23], and other drone wildlife videos from the web. We also collected drone footage in person at the San Diego Safari Park.

For the purpose of this project, we propose that our Jetson Xavier NX is onboard the Parrot Anafi and is connected to the internet throughout the flight (noted in Limitations). The “live stream” is coming from the drone’s camera and gets fed into our model on the Jetson, where inference is performed using the weights and hyperparameters of our best model. The video is broken into frames where bounding boxes will be predicted on. Each predicted bounding box with a greater than 50% confidence score will be sent to our cloud hosted web app. Inference is done using FP16 (16-bit floating point precision) to speed up inference time. During inference, we concurrently generate random latitude and longitude coordinates (noted in Privacy), the predicted class , the image of the predicted bounding box, and the confidence score. We bundle these metadata for every valid bounding box within each frame and send them together to avoid mismatching the predicted class and the frame or location that it corresponds to.

The message communication is facilitated by MQTT, a lightweight messaging protocol using a publisher/subscriber architecture for IoT devices [24]. Our messages were sent with a quality-of-service score of 1 to make sure the message is delivered at least once. The publisher lives on the Jetson, where it publishes the message bundles generated by our inference process. The subscriber lives on the cloud and listens to these bundled messages, decodes them, and updates our web app. Our web app is hosted by Flask which is also running on the same cloud server as the subscriber.

The update map process works by using Folium (a library used for visualizing geospatial data) [25] to display the corresponding icon of the predicted class (rhino, ostrich, giraffe, springbok, human) on the broadcasted location. The user can then click on the icon to see a popup which contains the frame with the predicted bounding box around the identified object together with its confidence score. Each time the user refreshes the page, the map is updated with the latest result pushed from the Jetson during inference.

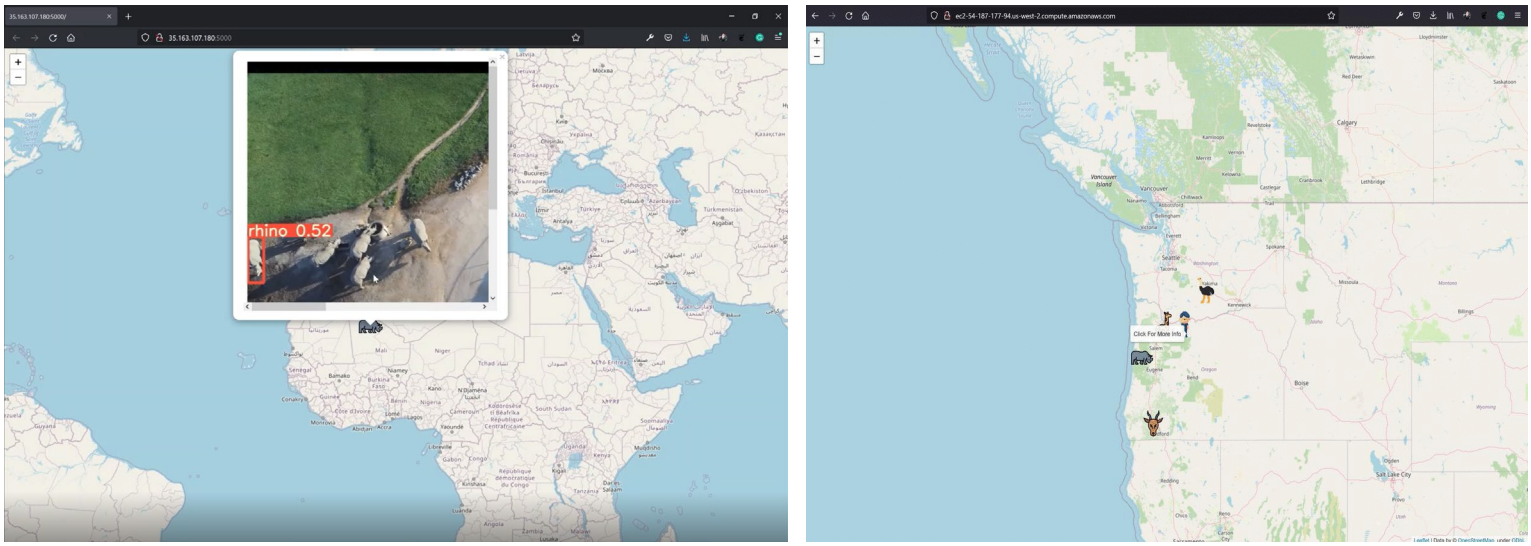

Figure 8: An example of a predicted bounding box (>50% confidence) on the broadcasted location

Limitations & Future work

1. **Starting from scratch + short time window = limited dataset:**
   The dataset was created from scratch and consisted mainly of videos provided by WildTrack and supplemented by videos and images from the wider internet. There was a lack of publicly available drone footage of wildlife. This led to a relatively small number of images and a limited number of classes which we could process and annotate. Having more images to train on will likely result in a model with better performance.

2. **Challenges in augmenting dataset**
   Besides applying online augmentation such as image flipping and blurring, we sought to increase the number of train images by using GANs and tiling. Due to time and compute constraints, we were unable to generate a large number of GAN images of sufficient quality, especially for classes such as springbok and ostrich, there are challenges with replicating their features with our limited dataset. Applying tiling helped enhance the dataset, as we were able to keep the features present within the larger images. However, this came at a very significant computation cost for training.

3. **Tiling for inference**
   Ideally our tiling would be used for training as well as inference. At inference, the incoming video stream would be broken up into individual frames and the frames would be tiled in memory. The tiles as well as the original frames themselves would be run through the YOLO model to generate predicted bounding boxes. Then, all of the predictions would be combined so that our original single input frame has a single set of predictions associated with it and we could have meaningful performance metrics for the input frames. In such a scenario, we determined that the ideal size for each tile would be 512x512 pixels. We did a study of distributions for our bounding boxes and found that the objects of interest tended to be less than about 200 pixels in a given dimension across all of our various input frame sizes. A 512x512 box would therefore minimize the chance of cutting the objects at the tile boundary and still maintain a large object to tile size ratio. Unfortunately, combining the predictions from multiple tiles together and doing all of the operations within memory proved too difficult to implement in our limited time frame. We ended up using tiling as an image augmentation process.

4. **Small object + shadows = distorted features and challenging bounding boxes**
   The majority of our images are made up of small objects with irregular outlines. The animals could travel in groups and be blocked partially by vegetation. These led to issues in discerning their outlines for accurate placement of bounding boxes. Attaining quality bounding boxes took up a significant proportion of the project time. We also often relied upon the shadows of the objects to second guess if there was an actual animal or person. Shadow recognition was not utilized in our project, but could be part of future work, functioning as a secondary detector. Shadow detection would be more complex, as we need to collect the shadows of all species across all times of day, at all possible angles from a drone (See Appendix Figure 13).

5. **A single model for 2 types of drones (quadcopter vs fixed wing)**
   Our train dataset was based on images from the lower-flying quadcopter (~30m) and the fixed wing drone which flies at a higher altitude (~70m). Images from these 2 sources were combined as they represented aerial imagery. We hypothesize that specialized models, one each for the quadcopter and the fixed wing, would allow the respective models to generalize better to their captured imagery, and lead to better model performance. We were unable to test this hypothesis, due to the limited quantity of images within our dataset which we could train the model on.

6. **Nighttime detection using infrared data + more urgent alerts + 4G live stream and cloud inference**
   Our model currently works during the day, however, poachers are most active at night. To achieve autonomous drone fleet detection, we could add infrared data for nighttime detection, and additional messaging system to send more urgent alerts than simply displaying predictions on our web app which requires user's attention to see the updates. Furthermore, the unmanned aircraft vehicle industry is moving toward cloud live streaming during flight, it is possible that our model can process live stream and infer on the cloud instead of the edge, in this case, we have to assume that the model is embedded to the drone software via our Jetson Xavier NX (see Appendix figure 14).

Future GAN Work

1. **Generate more images as synthetic dataset**
   As explained earlier, we tried using Photoshop to generate a synthetic image to train using SinGAN. Special care was spent to make sure that the image generated through Photoshop was as realistic as possible. However, per a Roboflow article [11] it is implied that naively pasting a target object into different backgrounds regardless of realism will have no negative impact on the training. That is, unrealistic images with target objects can still be used for training. We have found that image quality and quantity have a direct impact on performance, so it would be beneficial to synthetically increase the size of the training dataset.

2. **Improve SinGAN**
   We only used a limited number of SinGAN generated images for training due to the long training time and uncertainty in output image quality. There was a learning curve associated with selecting good training images to use that would generate usable output images. Even then, only a small subset of images were deemed usable.

   One possible way to improve both training time and output image quality would be to apply modifications to several mechanisms in the SinGAN architecture that can improve its training and generation capabilities [26]. Another possible way to improve SinGAN generated images would be to use an improved SinGAN model with an attention mechanism, specifically with relation to remote sensing images [27]. Per the referenced paper, the new method was able to generate better images with fewer layers when compared to the original SinGAN.

3. **CycleGAN and Contrastive Unpaired Translation (CUT)**
   Traditional image-to-image translation would have required a dataset composed of paired examples, where input image X (e.g. dry background) will need a corresponding expected output image Y (e.g. wet background) with the desired modification. Due to the way that our dataset was put together, it would have been impossible to generate this sort of one to one image comparison in the required environment.

   Alternatives to traditional image-to-image translation are CycleGAN [28] and Contrastive Unpaired Translation (CUT) [29]. These methods would allow us to generate image-to-image translation without having a paired dataset.

4. **Perceptual GAN & EESRGAN**
   One of the largest challenges to our model was the small object challenge, where our target objects were very small in relation to the overall size of the image. Two possible methods that address small object detection are Perceptual GAN [30] which super resolves representations of small objects, and edge-enhanced super-resolution GAN (EESRGAN) [26] which combines enhanced super-resolution GAN, edge-enhancement network, and a detection network to improve the quality of remote sensing images.

Security and Privacy

Due to the sensitive nature of our project (detecting and notifying an endangered species in the wild), we cannot publicly share our code base nor dataset, as poachers could utilize the same techniques to identify rhinos as we do. To maintain an edge in the arms race against the poachers, it is essential that our github repo and dataset are kept private. For the actual implementation, we would extract GPS coordinates from the video during inference to be sent to the cloud instead of generating random coordinates as we did in our project. We would also ask the user to log in to our web app to view where the detected animal is located and make sure that the detection information is sent to the web app in encrypted form. Authentication and encryption are necessary for the actual implementation.

Conclusion

We have demonstrated that small object wildlife detection from drone imagery is challenging but possible, even with a limited dataset which was built from scratch within the 6-week project window. We see how the use of techniques such as augmentation and regularization helped improve model performance, but at the cost of additional time and computation. We have also suggested methods to further improve model performance for the future. We highlight the limitations of drones and streaming information within a resource-poor setting.

Our current setup is a piece of a larger picture, where communication infrastructure and drone connectivity are essential. They need to come together, to allow for the model outputs to reach our target end-users. The end users need to act on that information to actually protect the rhinos. The technology is only effective if it is used as a part of a larger initiative where park rangers are able to deploy on the ground resources to intercept poachers and protect rhinos.

Rhinos face existential danger from habitat loss, climate change, and a black market for rhino horn which draws poachers from around the world. However, the situation is not hopeless. Wildlife reserves like Kuzikus in Namibia offer safe havens for the remaining rhinos and help combat habitat loss. Initiatives like WildTrack help these reserves track and protect these endangered animals from poachers. We hope that our project effort contributes to the larger conservation effort of protecting endangered rhinos by developing this proof of concept using available technology, with much luck, and a lot of care.

Appendix

Figure 9: Quality control examples

Figure 10: SinGAN super resolution

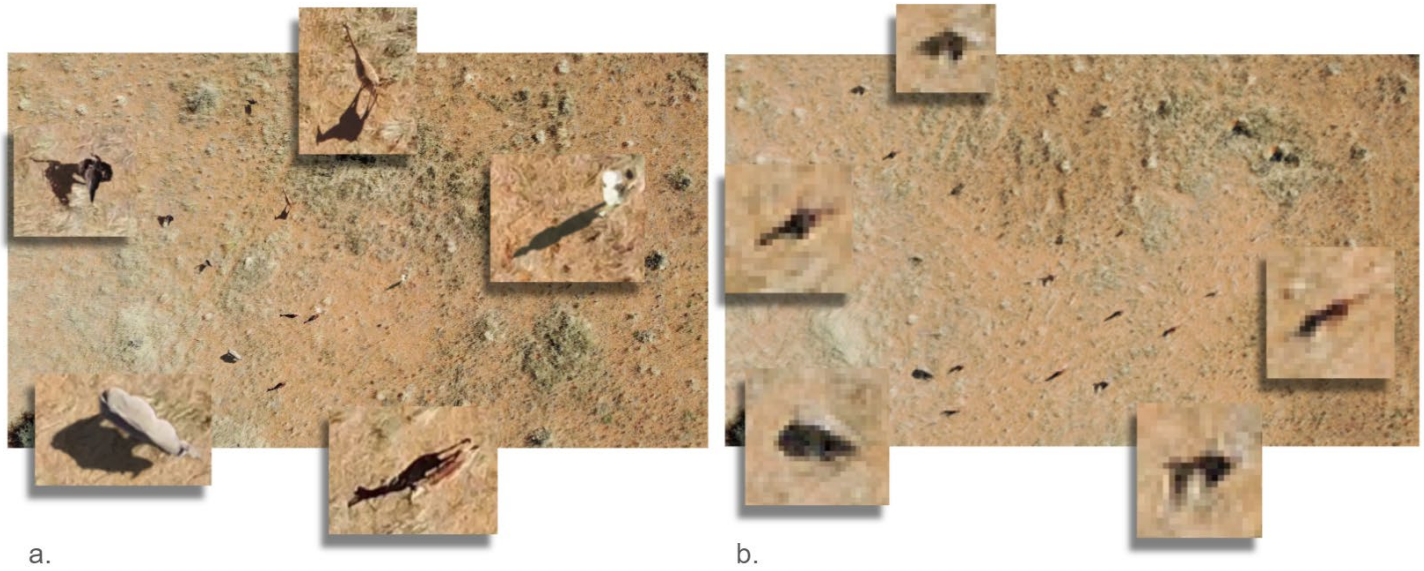

Figure 11: (a)Photoshopped input image with close up images to show details of objects and (b)SinGAN generated from input image with close up images to show details of objects

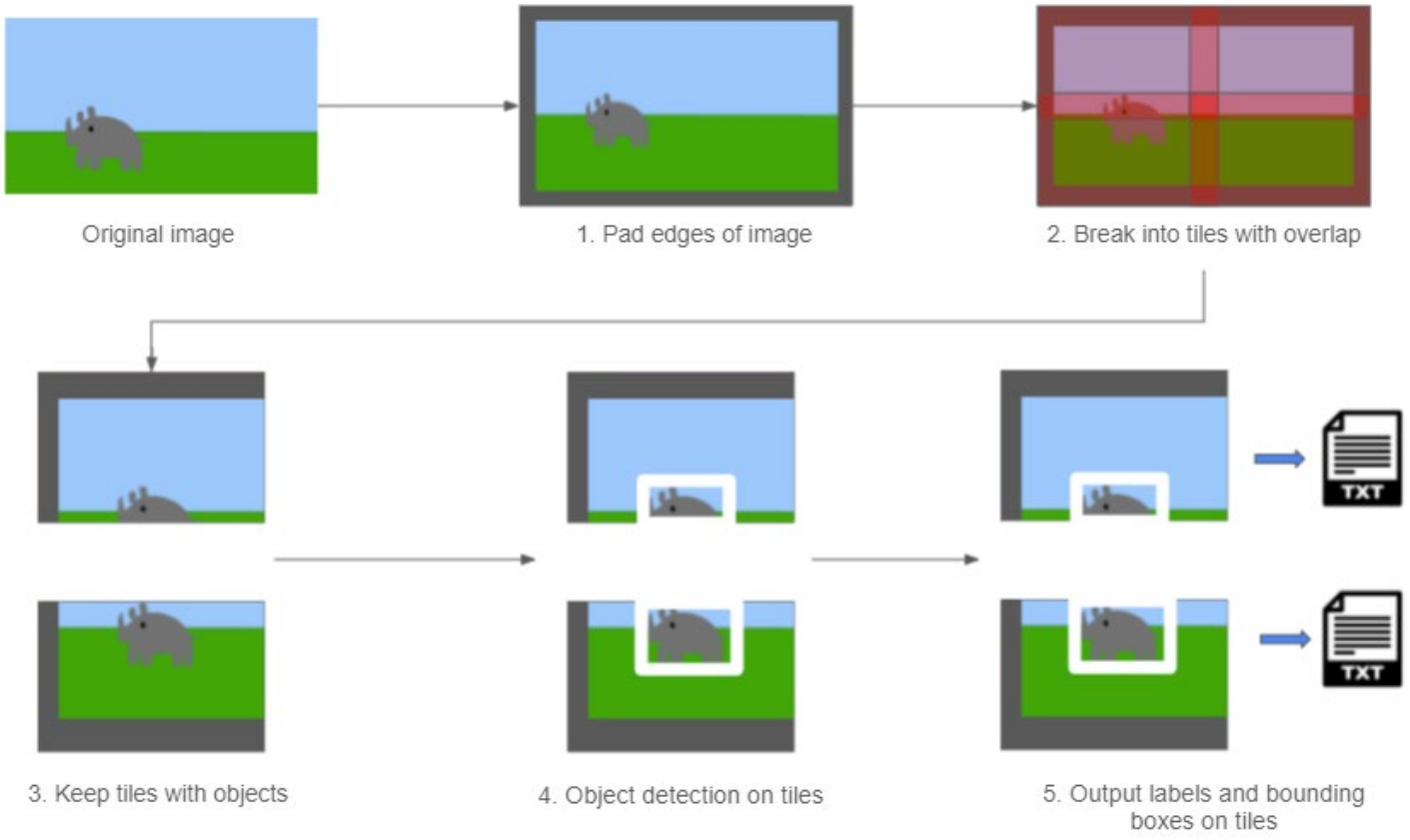

Figure 12: Step by step tiling process

Figure 13: Future work in shadow prediction - shadows are predictive of the animal class

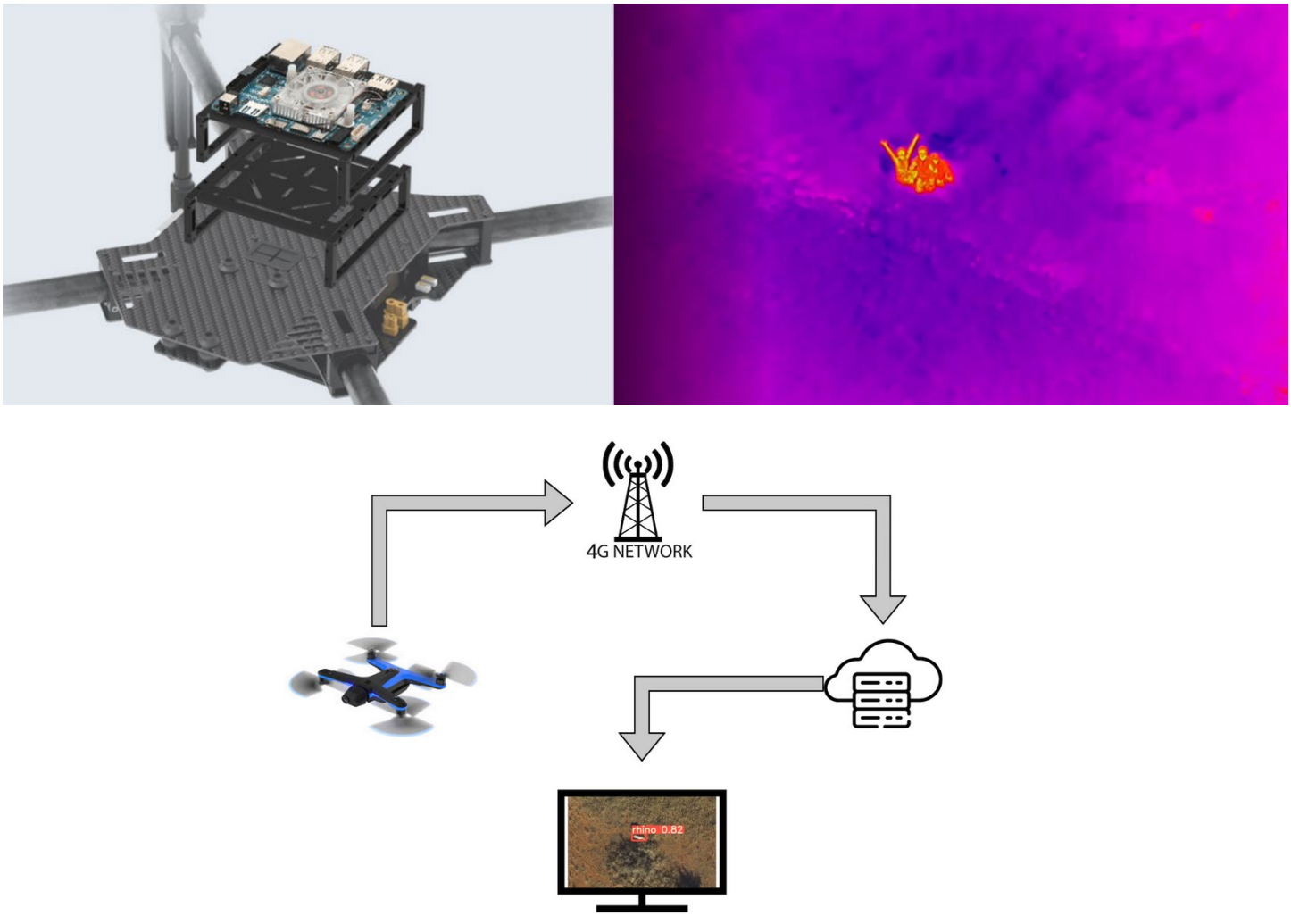

Figure 14: Future work with infrared data for nighttime detection & inference on the cloud [31]

References

1. https://www.worldwildlife.org/species/rhino
2. https://time.com/5482842/time-top-10-photos-2018-sudan-northern-white-rhino/
3. https://wildtrack.org/
4. https://www.parrot.com/us/drones/anafi
5. https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit
6. https://www.ischool.berkeley.edu/projects/2020/wildtrackai
7. https://www.csrwire.com/press_releases/45814-wildeyes-ai-helping-to-protect-wild-rhinos-from-poachers-and-track-species-recovery
8. Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Qinghua Hu, Haibin Ling. Vision Meets Drones: Past, Present and Future. arXiv preprint arXiv:2001.06303 (2020)
9. C. Kyrkou, G. Plastiras, T. Theocharides, S. I. Venieris and C. Bouganis, "DroNet: Efficient convolutional neural network detector for real-time UAV applications," 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 967-972, doi: 10.23919/DATE.2018.8342149.
10. F. Ö. Ünel, B. O. Özkalayci and C. Çiğla, "The Power of Tiling for Small Object Detection," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 582-591, doi: 10.1109/CVPRW.2019.00084.
11. https://blog.roboflow.com/detect-small-objects/
12. Shaham, T. R., et al. “SinGAN: Learning a Generative Model from a Single Natural Image.” ArXiv, 2019, https://arxiv.org/pdf/1905.01164.pdf.
13. https://github.com/ultralytics/yolov5/wiki/Tips-for-Best-Training-Results
14. https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/
15. https://github.com/ultralytics/yolov5
16. https://blog.roboflow.com/yolov5-improvements-and-evaluation/
17. https://pub.towardsai.net/yolo-v5-explained-and-demystified-4e4719891d69
18. Redmon, Joseph, et. al. “You Only Look Once: Unified, Real-Time Object Detection.” ArXiv, 2016, https://arxiv.org/pdf/1706.05274.pdf
19. https://blog.roboflow.com/what-is-an-anchor-box/
20. https://github.com/slanj/yolo-tiling
21. https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b
22. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173
23. San Diego Zoo still cam: https://zssd-kijami.player.camzonecdn.com/v1.5/CamzoneStreamPlayer?channel=zssd-kijami&iframe=yes
24. https://mqtt.org/
25. https://python-visualization.github.io/folium/index.html
26. Small-Object Detection in Remote Sensing Images with End-to-End Edge-Enhanced GAN and Object Detector Network. Jakaria Rabbi, Nilanjan Ray, Matthias Schubert, Subir Chowdhury and Dennis Chao, arXiv:2003.09085 [cs.CV] (2020)
27. Gu, S., et al. “Improved SinGAN Integrated with an Attentional Mechanism for Remote Sensing Image Classification.” Remote Sensing, 2021, https://www.mdpi.com/2072-4292/13/9/1713/htm.
28. Zhu, J., et al. “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks.” ArXiv, 2020, https://arxiv.org/pdf/1703.10593.pdf.

29. Park, T., et al. “Contrastive Learning for Unpaired Image-to-Image Translation.” ArXiv, 2020, https://arxiv.org/pdf/2007.15651.pdf.
30. Li, J., et al. “Perceptual Generative Adversarial Networks for Small Object Detection.” ArXiv, 2017, https://arxiv.org/pdf/1706.05274.pdf.
31. https://www.parrot.com/us/drones/anafi-ai/technical-documentation/connectivity