Body-Part Enabled Wildlife Detection and Tracking in Video Sequences

Alberto Lee[1], Kofi Appiah[1][a] and Sze Chai Kwok[2][b]

[1]*Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, U.K.*

[2]*Phylo-Cognition Laboratory, Division of Natural and Applied Sciences, Data Science Research Center, Duke Kunshan University, Duke Institute for Brain Sciences, Kunshan, 610101, Jiangsu, China*

{*al1414, kofi.appiah*}*@york.ac.uk, szechai.kwok@duke.edu*

Keywords: Animal Detection, Deep Convolutional Neural Networks, Real-Time Tracking, Data Augmentation.

Abstract: Tracking wild animals through videos presents a non-intrusive and cost-effective way of gathering scientific information key for conservation. State-of-the-art research has shown convolutional neural networks to be highly accurate, however, the application of this field on wild animal tracking has had relatively little interest. This is potentially due to the challenges of varying illumination, noisy backgrounds and camouflaged animals intrinsic to the problem. The aim of this work is to explore and apply state-of-the-art research to detect and track wild animals (specifically bears and primates, including their body parts) in video sequences in real-time. Due to obstructors such as foliage being prevalent in wild animal environments, body part tracking presents a solution to detecting animals when they are obstructed. Two deep convolutional neural networks (YOLOv4 and YOLOv4-Tiny) are trained to detect and track animals in their natural habitat. By using the knowledge that an animal is composed of body parts, the score of weakly predicted bounding is boosted from the relative distance of related body parts. For tracking, the K-Means algorithm is used to locate the average position of each animal in frame. With the introduction of a body-part confidence boosting, the detection rate can be increased by approximately 2% for a weakly predicted class.

1 INTRODUCTION

Animal tracking has been done previously with GPS tracking, radio-tracking and other methods (Waldmann et al., 2022). These have effectively collected positional data, mapping migration routes and feeding areas. However, manual capturing and tagging of animals are known to be difficult, time-consuming and potentially harmful. The emergence of camera traps is continually collecting large quantities of animal data into datasets although large datasets can take many months to analyse (Swanson et al., 2015). The solution to this problem lies within the field of Computer Vision which encompasses the techniques allowing a computer to attain a high-level understanding of digital images.

Wildlife is under threat. According to the World Wide Fund for Nature, an estimated 10,000 species are going extinct every year. The average rate of species loss is a hundred times higher than in the previous century. Indicating the world is now in its 6th mass global extinction event (ceballos et al., 2015). Advancements in using vision algorithms to track wild animals provide useful insights into a species population (Rowcliffe et al., 2008) and nature. By automatically detecting animals, considerable time and resources can be saved while warning researchers about wildlife threats. This work focuses on :

- Understanding the effectiveness of a wild animal tracking algorithm.
- Understanding how animal body-part detection can be utilised.
- Understanding how state-of-the-art object tracking methods can be effectively applied to the medium to achieve the best possible results.
- Understanding how an animal can be tracked throughout a video sequence.

The rest of the paper is organised as follows, section 2 describes related work. This will be followed by details of our proposed approach in section 3 and how it has been implemented in section 4. The experimental results and discussions will be presented in section 5 and followed by the conclusion and future work in section 6.

[a] https://orcid.org/0000-0002-9480-0679

[b] https://orcid.org/0000-0002-7439-1193

2 RELATED WORK

For several years, convolutional neural networks (CNNs) have been the leading algorithm in terms of accuracy of object classification. None of the known CNN architectures is explicitly designed for tracking wild animals, although due to the nature of CNNs, they can learn to classify new images when given the right dataset. Classification methods have been able to achieve great accuracy on smaller datasets with much less computational power. (Ciresan et al., 2012) have been able to achieve human-like error rates on labelling handwritten numbers. Objects found in realistic settings pose a greater challenge due to their high variability. Natural scenes such as forests and animals contain much more noise and variance compared to urban areas. (Wang et al., 2020) proposed a context-aware CompositionalNets, which increases the detection performance on strongly occluded vehicle but not necessarily animals in the wild.

In (Villa et al., 2017), the performance of both GoogLeNet and VGGnet was tested. They were trained on the Snapshot Serengeti dataset (Swanson et al., 2015) to classify the 26 most common species and evaluate their accuracy. They found that these models had an average error of 19.38% and struggled when part of an animal's body was showing. Meaning there was not enough information to classify the animal from detected body part. This raises a question - could identifying individual body parts (body-part boosting) enhance the accuracy of a model?

Natural environment animals inhabit poses some of these technical challenges to vision algorithms:

- Varying illumination: A gradual shift of light.
- Noisy backgrounds: Due to the swaying of vegetation and varying weather conditions.
- Animal camouflage: Resulting pixel values can be very similar to the background.
- Shadows: A moving object's shadow.

It is difficult to use unsupervised methods for object detection within these scenes as most of these methods, including optical flow and frame differencing are very sensitive to noise. The highly dynamic and cluttered background of natural environments calls for more advanced background subtraction techniques. (Ren et al., 2013) used a combination of foreground object segmentation graph cutting and temporal information with a fusion of neighbouring frames to detect objects in videos. With a focus on dynamic backgrounds, including forest landscapes, they achieve high precision on foreground segmentation by effectively using the temporality of videos.

(Gomez et al., 2016) achieved an accuracy of 88.9% over the Serengeti dataset using very deep CNNs. By testing six state-of-the-art networks at the time, they displayed that deep networks outperformed shallow networks on the majority of animal classes. The CNN models were pre-trained on the ImageNet dataset and then trained over their animal-specific database. They used the pre-trained CNNs under the assumption that models would have already learnt the basic features for general image classification. It is unclear if the accuracy would be negatively affected if the models were trained from scratch on the Serengeti dataset. Unlike (Ren et al., 2013), it is highlighted that they did not take advantage of the temporal dimension of the bursts of images captured by the camera.

3 OUR APPROACH

The overall aim is to create a real-time framework for detecting, tracking and classifying specific wild animals, including body parts in video sequences. There is no hard limit on what defines a real-time detection speed. We define real-time to be a frame-rate where an animal's movement appears continuous, with no visible time lags. For this definition, we aim for a frame-rate of at least 20 FPS.

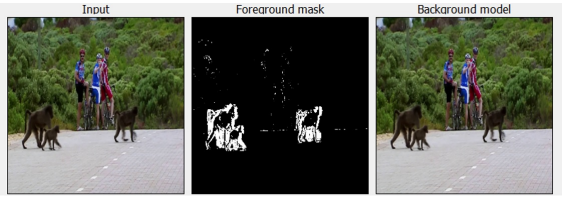

Figure 1: A figure showing the results of background subtraction as applied to a sample video.

3.1 Object Localization Method

The aim is to evaluate a number of techniques against the expected outcome of this work. Thus, the algorithm must give a real-time performance on our test data and hence, algorithms like optical flow would be too slow to consider. By testing Mixture of Gaussian (MoG) and other algorithms on sample videos taken from our dataset (Wang et al., 2020) using a simple background subtraction (Sobral et al., 2013) technique, a visual inspection seemed promising at first as shown in figure 1. However, when the input data is taken from a moving camera, the animal would be misclassified with background differencing and will require a more expensive technique like optical flow.

CNNs allow for accurate object detection if given enough relevant training data. While many CNNs do not run in real-time (Girshick, 2015), the recent YOLO-v4 algorithm does and the architecture will be used for classification in this work. Complex backgrounds are less of a problem with CNNs compared to other methods. Thus, allowing them to adapt to many different backgrounds. The YOLO algorithm will also not struggle with a moving camera as it runs on each frame compared to background differencing algorithms. YOLOv4-Tiny, a small variant of YOLOv4 (Bochkovskiy et al., 2020), which significantly enhances the processing speed will be used for localisation in this work.

3.2 Dataset Selection

There are limited publicly available labelled animal video datasets and hence, in this work datasets from multiple sources have been utilised. All video data collected with animals have been divided into training data and validation data, just to ensure our models generalise well after training. The test data has been generated from the Kwok-Lab dataset (Wang et al., 2020). The Kwok-Lab dataset consists of 2000 video clips, each about $4-6$ seconds long taken from various sources from television programs; thus approximately $200,000$ images with various animals. While both papers do not have relevance in the field of computer vision, the dataset has been used in this work.

Figure 2: Sample images of chimps used for testing.

The dataset is separated into two halves. One half contains clips showing one or more primate, specifically different monkey species as shown in figure 2. The other half is of other types of mammals like bears (as shown in figure 3), and phyla such as reptiles, fish, birds and insects. Majority of the images from the video data have been used as validation dataset to analyse how well our approach fulfils its objectives. Performance analysis has been conducted both visually and quantitatively. To gather quantitative performance data, two videos from the Kwok-Lab dataset are annotated with ground truths. Each video contains at least a bear or a primate.

Figure 3: Sample images of bears used for testing.

By visual inspection, it has been established that the Snapshot Serengeti database (Swanson et al., 2015) contains many of the same animals as the Kwok-Lab dataset. The Snapshot Serengeti database is also widely used for training CNNs to detect animals (Villa et al., 2017); a good starting point when looking for a suitable architecture for transfer learning. Because the Snapshot Serengeti database isn't easily accessible, we have also used a more accessible dataset from Kaggle with about 30 different species. We have tested our approach on species common to all the datasets and conducted tests on bears and primates as they both have special characteristics suitable for our proposed system:

- they have a more natural backgrounds in their videos
- they both share similar (but not identical) anatomical features
- a good number of CNN models exit which have been pre-trained on the COCO dataset (Lin et al., 2014) with images of bears but none of primates.

The choice of bears and primates makes it easier to separate the body parts into the four distinct classes needed to train our model; head, arm, body (or torso) and full (the entire animal).

4 IMPLEMENTATION DETAILS

We trained a Mask R-CNN (He et al., 2017) on a small manually annotated dataset of primates and bears where each body part is segmented using polygons, a pipeline for the training is as shown in figure 4. The open-source annotation program LabelMe (Wada, 2016), was used in the manual annotation and a sample output is as shown in figure 5. The trained

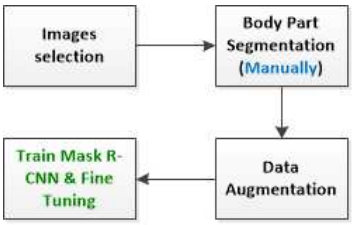

Figure 4: The semi-automated data generation pipeline to train the body-part network.

algorithm (Mask R-CNN (He et al., 2017)) automatically segments classes of interest on new data, similar to the manual labelling of these classes. This semi-automation yields a method for rapid labelled dataset creation and offers a course background subtraction of each body part; an enhancement to the mask R-CNN model. With just over 50 manually annotated images for each specie the mask R-CNN model (Abdulla, 2017) was suitably trained to detect the body parts after data augmentation.

The Mask R-CNN (He et al., 2017) model presented in (Abdulla, 2017) is pre-trained on the COCO dataset so we trained a selected number of classes (eight in this case) on the last network layer with a high learning rate of 0.001, freezing all other layers so previously learned features are not destroyed. An additional fine-tuning step is used, this is possible because mask R-CNN is built of the R-CNN architecture meaning it's a single-state pipeline. After 5 epochs all layers are unfrozen and trained with a small learning rate. We use a learning rate of $1/10^{th}$ of the prior learning rate. This small learning rate allows the model to fine-tune to our custom classes without strongly influencing the pre-trained layers. The model is fine-tuned for 5 more epochs. Ten percent (10%) of the manually annotated data is used for validation.

The trained model can segment body parts with reasonable accuracy. By accurately segmenting sections of the class we can simulate obstruction of body parts that may be caused by the foliage of forest landscapes. To enhance the extraction of body parts from each image and also avoid the inclusion of pixels outside the object (animal in this case), a custom background removal function is used. The function takes the trained model and a list of images as inputs, it runs over each image, predicting a list of masks. All alphas of pixels outside the masks are set to zero and the image is then cropped down to the limits of the masks. Each image is saved with the name of each predicted class and its associated bounding box. We then split the dataset into a training and test set, keeping a ratio of $80:20$. The extracted image classes had to be cleaned manually by removing indiscernible and wrong class predictions that would negatively affect model training as shown in figure 6. Finally, a series of background images are collected from FreeImages consisting of varied forest and urban landscapes. We chose to include urban backgrounds as some videos from the Kwok-Lab dataset contained urban areas. A custom data creation function is used to augment the cropped masks onto these backgrounds, taking the mask images and backgrounds as inputs to generate the new image as shown in figure 7.

4.1 Animal Tracking

There are many ways to represent an animal's movement, for example the use of directional arrows or explicit text descriptions. We use directional arrows as this specifies the direction of movement and allows for a full 360 degrees movement. First, to find the course position of each animal we represent each predicted bounding box as a single position by locating the middle coordinate. We then use $K-Means$ clustering on these coordinates; where the number of clusters equal to the number of detected whole animals (bear or primate in this case), so a series of cluster-centres are generated. The cluster-centres are used to track each animal as it moves through the video, however, these clusters are sporadic and hence not used as the location for the directional arrows. Instead, we use a series of *"moving-points"* that gradually follow the clusters. Each *moving-point (mp)* consists of a coordinate and a directional unit vector. The moving-points, function under two rules. Rule one is to create a new *mp* when a cluster has no *mp* within a specified radius of r. The new point is assigned the position of the cluster. The second rule is that for any *mp*, if there are no clusters within r then the *mp* is deleted. These rules allow the *moving-points* to adapt to all animals in a frame, dealing with the cases where animals enter and leave the scene. To prevent sporadic movement each *mp* moves towards clusters within radius r with a predefined weight as in Equation 1.

$$mp_{t-1} = (cluster_{centre} - mp_t) * weight \qquad (1)$$

A space of previous points is kept allowing the algorithm to keep a history of the movement. For each cluster, a series of *local-points* are located by searching the movement history and keeping points that are located within a radius of the cluster. A look-back value is set, LB, that forces the *local-points* to ignore the most recent LB movement history entries. This allows for a larger gap in time between the cluster and where previous clusters have been located. Thus causing the *moving-points* to smoothly follow the clusters reducing sporadic and local movements. The mean of

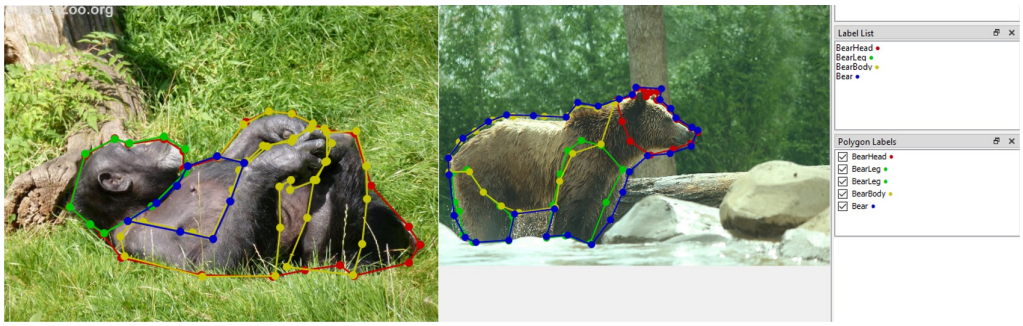

Figure 5: Sample output of the segmented parts of a bear and primate using LabelMe.

Figure 6: An image of a primate misclassified as a bear.

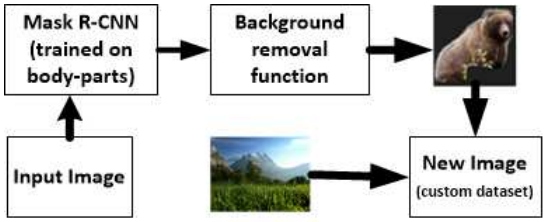

Figure 7: A pipeline for generating our custom dataset.

all unit vectors *uv* from each *local-point* to the current cluster is calculated as μ_{uv}. For each *mp* within a predefined radius of the cluster, the unit vector of each *mp* is calculated as the weighted sum of itself and μ_{uv}, which is then normalised as a unit vector. Finally, to visualise the movement direction, for each *mp*, an arrow of set size is drawn in the direction of the respective unit vector and is centred on the respective coordinate. This method produces a minor computational overhead that decreases detection speed by $1-2$ FPS.

To make use of the body-parts, we use the fact that an animal's body parts are always close to each other, hence for each predicted body-part, its confidence score can be increased if other body parts are nearby. Using this theory, each predicted-box (p), is checked and if the score is lower than a set threshold it is selected for confidence boosting (body-part boosting). This targets only boxes with low scores, preventing the problem that if all boxes were boosted, the relative scores would cancel each other out. Next, for each box of the same class or related animal part classes (s), and the Intersection over Union (IoU) between s and p is calculated. In which the overlap between the predicted and ground truth bounding boxes is known as the IoU. Given that the IoU is greater than a set threshold, the score of p is incremented by a set value, where the threshold represents whether each class is close enough to be considered part of the same animal. This is conducted as an extra pre-processing step so certain boxes are held more favourably for Non-Maximum Suppression (NMS).

5 RESULTS & DISCUSSION

In testing we vary the input resolution, architecture, training data, body-part confidence boost and Non-Maximal Suppression (NMS) method. A larger input resolution can help the model detect small objects. However, it also increases the amount of GPU memory required, causing the model to fail if memory is limited. The input resolution also impacts how fast the model can run and hence we test lower resolutions to analyse the trade-off between speed and detection accuracy. We test input resolutions of 416, 320 and 160. We test the YOLOv4 algorithm against its smaller variant (YOLOv4-tiny), because of its exceptional detection speeds. We assumed that augmented training data will help improve detection accuracy. To justify this we test the network separately with augmented training data and without augmented training data. We also test the original dataset used for training the mask R-CNN algorithm to demonstrate the effects of automatic bounding box creation.

We compare the effectiveness of soft-NMS as its impact on computation is negligible and it is gives an overall improvement on prediction accuracy. For transfer learning, we use weights pre-trained on the COCO dataset. We chose weights trained on this dataset as it includes many animals (such as bears) and the animal images contain their natural environment. The inclusion of bears in the training data allows for a comparison between primates and bears to display the benefits of transfer learning. We decided on 60 epochs for training as this achieves satisfactory results within a reasonable time (see figure 8). For the final model, we trained for 100 epochs to ensure the best results possible. As YOLOv4 uses a custom loss function we do not change it as this would most likely damage the performance. Larger batch sizes improve convergence and accuracy, so a batch size of 8 is used to fit any memory constraint GPU.

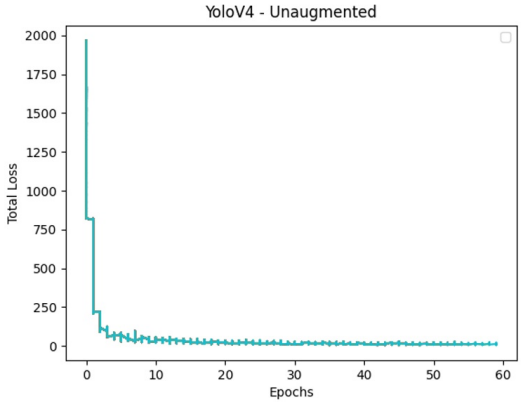

Figure 8: Total loss against epochs for YOLOv4.

By way of robustness, if the IoU is greater than a set IoU threshold the prediction is classed as a true positive else, it is a false positive; this measures the precision of detection. We set the IoU threshold to 0.45 which is more conservative with less precise predictions and it is chosen with the reasoning that the selected classes are very challenging.

It is significantly harder to detect body parts compared to the entire animal. The most distinctive body part, the head, performed better than all other body parts for both primates and bears. We assume this discrepancy is due to the ambiguous nature of these classes as they lack distinctive features. With this reasoning it is clear to see why the 'Body' class for each animal was the worst-performing feature. Table 1 shows the precision of detection for primate arms and bodies are less than the other classes. Combined with prior reasons, comparing these classes to bear legs and bodies we notice a significant difference in accuracy which is likely due to the model's use of transfer learning. Since the model adopts weights that have been trained on a much greater dataset of bears, it is logical to assume the model is more useful to detect all aspects of bears. For the video set, it must be noted that there is a limited amount of environmental variation due to a single environment shown in each video. Meaning the algorithmic performance may not always be true for all videos. The model has a detection speed of 15.77 FPS achieved on a Tesla V100 GPU. The mean average precision (*mAP*) value of 60.1% is worse than 65.7% as reported for the original YOLOv4 trained on the COCO dataset. As the two values are relatively close, it demonstrates that our implementation can improved further to match the original 65.7% reported.

5.1 Impact of Architectures

From testing the YOLOv4 variant, YOLOv4-Tiny with the baseline model, we noticed FPS improvements of 20.92 (+5.15). An increase was expected however, the reported Frames per second (FPS) improvement reported in (Bochkovskiy et al., 2020) is approximately eight times that of YOLOv4. The difference in hardware might have contributed to this discrepancy. It was assumed that the performance would be affected as YOLOv4-Tiny is less powerful, yet the decrease in precision was very minimal as shown in figure 9. For the video dataset, it significantly outperformed the baseline for some classes like primate, bear head and bear leg as in figure 10.

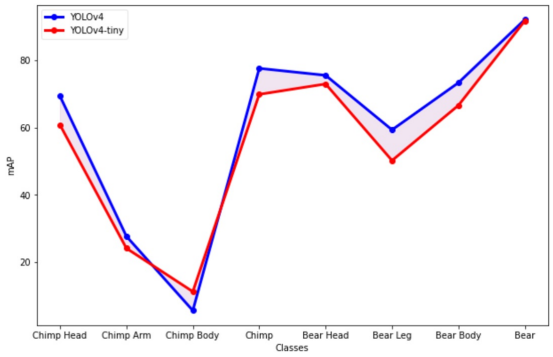

Figure 9: Test set architectural difference.

We argue that where this architecture performs worse than the baseline the discrepancy is less important as it only performs worse on areas that already have high precision. The model should predict all classes with reasonable accuracy rather than a few with high accuracy. The video set is not representative of all scenarios and the improvement in FPS cannot be understated.

Table 1: Results of the baseline model, where P. is Primate and B. is Bear.

	P. Head	P. Arm	P. Body	Primate	B. Head	B. Arm	B. Body	Bear
Test Set(AP%)	69.337	27.686	5.625	77.619	75.545	59.339	73.347	92.137
Video Set(AP %)	79.543	12.944	0.735	48.501	63.285	22.260	95.238	98.158

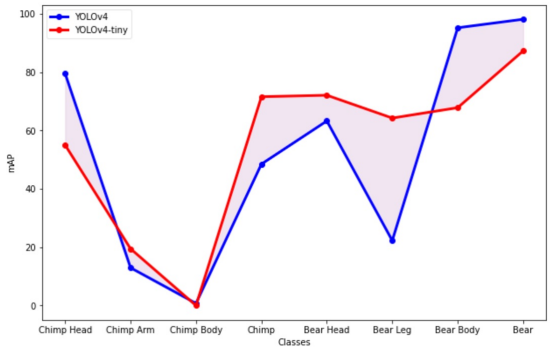

Figure 10: Video set architectural difference.

5.2 Best Model

This model uses the YOLOv4-Tiny algorithm (final solution), an input size of 320, augmented auto-generated training data and Soft-NMS. Due to the inconclusive results on the influence of architecture, we test the same model as described but using the YOLOv4 algorithm (baseline) for comparison. Each network was trained for 100 epochs or until it began to over-fit.

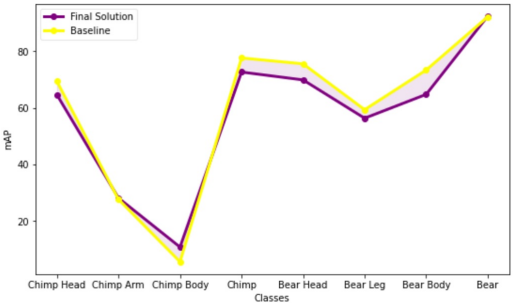

Figure 11: Performance of YOLOv4 against our best implementation using the test set.

The comparison between different architectures shows very similar results. The solution with YOLOv4 achieves a slightly higher *mAP* of 62.87% (+4.68%) on the test set. Yet the solution with YOLOv4-Tiny achieves a significantly higher detection speed of 23.25 (+10.24) FPS. With a speed that satisfies the constraint of real-time, this solution is the best model for any resource constraint hardware. A comparison against the baseline shows an improvement using the YOLOv4-Tiny architecture relative to previous tests as shown in figures 11 and 12. These tests have optimised the baseline to best fit the

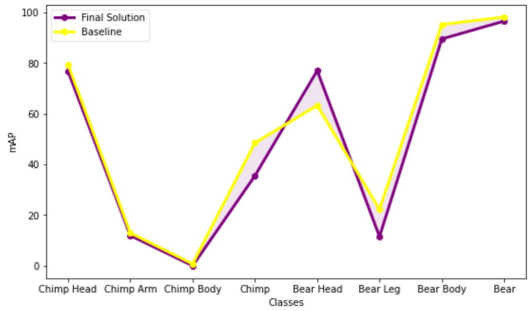

Figure 12: Performance of YOLOv4 against our best implementation using the video set.

goals of this work. The best model was evaluated using a series of videos similar to those shown in figures 13 and 14. In general, the detected classes appear correct with the head, body and full animal being the most representative of the truth. While the arm and leg classes are often correctly detected, the nature of the animal walking causes the bounding boxes to constantly change shape and position. The model struggles when two animals are very close together, predicting both animals as one, or when there is a large amount of obstructing foliage.

Figure 13: A sample output from our proposed model with labelled parts of a chimp.

Upon adjusting the tracking algorithm hyper-parameters, setting the radius for both *moving-points* and *local-points* to 150; the coordinate and vector weights set to 0.1, allowed for smooth movement and directional adjustment. A large *lookback* (LB) value will cause the arrows to be more delayed and yet appear smooth. We settled on a LB of 30, roughly equating to one second of video, allowing a sufficient

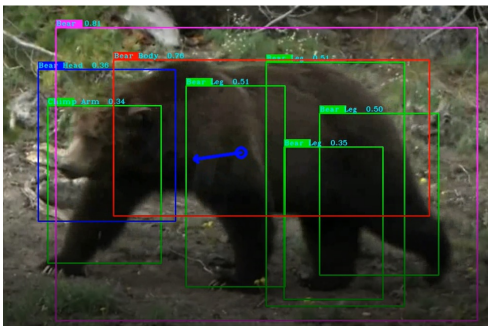

Figure 14: A sample output from our proposed model with labelled parts of a bear.

amount of time for animals to show a clear movement in direction. Movement tracking, while taking a second to begin (due to LB), effectively shows the direction of movement for each animal in the frame. Each arrow was always located on the animal and moved with the animal at a relatively constant rate. The arrows turn to find the direction of movement and adapt if the animal changes direction. A caveat to this is that the camera had to be near stationary, as a moving camera meant the algorithm is tricked into thinking the animal is moving in the opposite direction.

6 CONCLUSION

This paper presents solutions to many of the intrinsic problems associated with wild animal tracking. There are many facets for future development that will not only improve the accuracy and usability, but also other potential applications. The approach is capable of detection animals at 23.35FPS, just 0.65FPS short of a classical movie. This is noticeably smooth, demonstrating greater speed and accuracy than unsupervised object detection techniques like Mixture of Gaussian's. In general, current state-of-the-art methods for tracking in videos proved effective, however, visual inspection showed predictions sometimes lacked clarity, appearing sporadic and unstable. One reason for this is that the methods do not harness the extra temporal information supplied by the video format. An avenue of research could use the Sequential-NMS algorithm as a post-processing method to give higher confidence on low confidence predictions.

REFERENCES

Abdulla, W. (2017). Mask r-cnn for object detection and instance segmentation on keras and tensorflow. *GitHub repository*.

Bochkovskiy, A., Wang, C., and Liao, H. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934.

ceballos, G., paul r. Ehrlich, anthony d. Barnosky, andrés garcía, robert m. Pringle, and todd m. Palmer (2015). Accelerated modern human–induced species losses: Entering the sixth mass extinction.

Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *Arxiv preprint arXiv:1202.2745*.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Gomez, A., Salazar, A., and Vargas, F. (2016). Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks. *arXiv preprint arXiv:1603.06169*.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Ren, X., Han, T. X., and He, Z. (2013). Ensemble video object cut in highly dynamic scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1947–1954.

Rowcliffe, J. M., Field, J., Turvey, S. T., and Carbone, C. (2008). Estimating animal density using camera traps without the need for individual recognition. *Journal of Applied Ecology*.

Sobral, A. et al. (2013). Bgslibrary: An opencv c++ background subtraction library. In *IX Workshop de Visao Computacional*, volume 27.

Swanson, A., Kosmala, M., Lintott, C., Simpson, R., Smith, A., and Packer, C. (2015). Snapshot serengeti, high-frequency annotated camera trap images of 40 mammalian species in an african savanna. *Sci Data 2:150026*.

Villa, A. G., Salazar, A., and Vargas, F. (2017). Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks. *Ecological Informatics, Volume 41*.

Wada, K. (2016). labelme: Image polygonal annotation with python. Github.com.

Waldmann, U., Naik, H., Máté, N., Kano, F., Couzin, I. D., Deussen, O., and Goldlücke, B. (2022). I-muppet: Interactive multi-pigeon pose estimation and tracking. In *Pattern Recognition*. Springer International Pub.

Wang, L., Zuo, S., Cai, Y., Zhang, B., Wang, H., Kwok, S. C., et al. (2020). Fallacious reversal of event-order during recall reveals memory reconstruction in rhesus monkeys. *Behavioural Brain Research*, 394:112830.