Exploring Sequences for Improving Camera Trap Image Classification

Yash Kale*
yashkale@uw.edu

Darshan Mehta*
darshanm@uw.edu

Bhuvan Malladihalli Shashidhara*
msbhuvan@uw.edu

Dan Morris (Sponsor)
dan@microsoft.com

*Equally contributing authors

1 Motivation

Camera Traps are extensively used to observe wildlife in their natural habitat. A camera trap is a remotely activated camera that is equipped with a motion sensor or an infrared sensor, or uses a light beam as a trigger. Camera trapping is a method to capture wild animals on film without disturbing the ecosystem, and has been used in ecological research for decades [21]. This could help in the early detection of natural or human threat to animals, and help towards ecological conservation.

Currently, a massive number of such camera traps have been deployed at various ecological conservation areas around the world, collecting data for decades. Wildlife researchers are interested in performing ecological studies like population estimation of different animal species across different seasons of a particular ecological conservation area. But, due to the vast size of data, it is often very difficult for researchers to go through all the images taken by the camera traps manually to perform such studies. Automating the process of labeling camera trap images has had a high impact on enabling wildlife research at such scales and as a result, it has been an active area of research in recent times.

The AI for Earth group at Microsoft [12] has developed the MegaDetector [6], and the Microsoft AI for Earth Species Classification [7] software, which uses deep learning to identify the animal species present in a given image. These models are trained on various open-source camera trap images and have achieved great results. These systems, however, perform classification on a single image. Due to the camouflaging capability of animals in their natural habitat, it can sometimes get difficult to identify and classify animals from merely a single image. This problem gets particularly challenging during night time, for instance - if a rabbit is found at night, we only see two bright dots (rabbit's eyes) in a single image, but over a sequence of images we can see these bright dots move in a pattern which enables us to identify it as a rabbit. We hypothesize that on observing subsequent frames of a camera-trapped video, assuming that the animal moves, it gets much easier for a human as well as a machine to locate and classify the animal. Hence, in this research project, we explore and measure the impact of using image sequences on improving the camera trap image classification.

The paper is organized by describing the dataset used in the research, followed by the problem statement and objectives. Further, we discuss few prior works related to our problem. This is followed by detailed description of proposed methodology, and results.

2 Dataset Description

The Wellington dataset [1] comprises of 90,150 sequences collected from camera traps across 187 locations in Wellington, New Zealand. Each sequence consists of 3 images 1 leading to a total of 270,450 images in the dataset consuming 190 GB disk space. Each of the sequences have been hand-labeled into one of the 17 classes: 15 classes corresponding to various animal species, the empty class for sequence that contain no animals, and the unclassifiable. This dataset is listed on Lila BC [2] which internally uses Microsoft Azure to host the data. All the image labels in all the dataset are maintained in COCO Camera Traps Format [11]. The above datasets are open for public non-commercial research purposes, and their exact licenses can be found in respective reference links. Labels in the dataset are given at a sequence level i.e each image within a sequence has the same label. In other words, a burst or sequence of 3 images would be treated as a single data point and would have a corresponding label. Data processing steps and further details regarding the data and challenges posed by it are available in the section 5

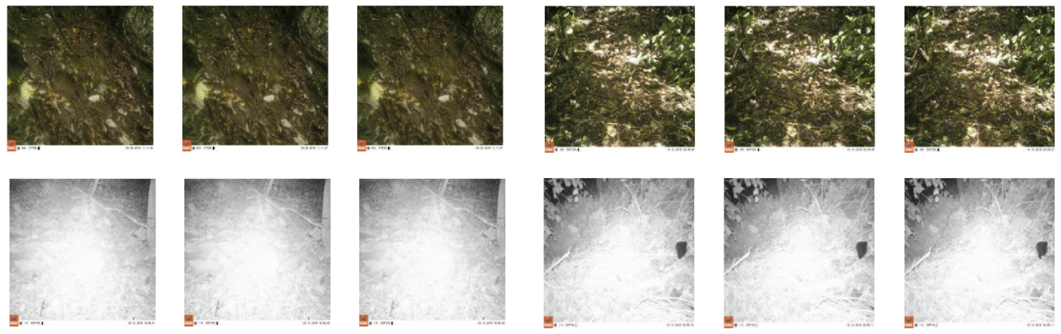
Figure 1: Four sequences from Wellington data showing the burst of three images each.

3 Problem Statement and Objectives

In this project, we focus on a binary classification task to predict whether a sequence of images (burst of 3 images), contains an animal or not. In order to determine if the sequence information helps, we consider the following two categories of models:

- **Baseline Models**: Single-image-based models which learn from a single-image input (single image from the burst of images), and does not utilize any sequence information. These models represent the currently adopted strategy in many systems.
- **Sequence Models**: Models which utilize sequence information from the burst or sequence of images in some way during the training and inference processes.

Our main aim is to determine if the addition of sequence-information that is present in the image sequences help in improving the image classification process. In other words, we aim to test if the Sequence Models perform better than the Baseline models on the test set. We use Area under ROC [22] as our primary evaluation metric for measuring the performance of the image classification models.

4 Background and Literature Review

Wei et al. [20] built a tool named 'Zilong' that uses a purely image processing based approach to identify empty images in a camera trap dataset. They use pixel level differences between frames to measure color change and Sobel algorithm to identify edges. They have a slightly separate pipeline for foggy vs. non-foggy images. While this approach is extremely fast due to the simplicity of the computational steps, it performed poorly in scenarios where empty images contained swinging vegetation since these are usually associated with a color intensity and edge change.

Unlike Wei et al. [20], Yousif et al. [24] built a software tool named 'Animal Scanner' which uses both computer vision algorithms and deep learning to detect and classify moving objects. Similar to us, they mainly focus on aiding the removal of empty frames in camera trap images. Their predictions fall into three categories: animal, human, and background. They use the information available in the sequences for background subtraction thereby identifying regions which could potentially contain a moving object. This helps in removing focus from the heavy vegetation present in most scenes in the wild. They then reduce the false positive foreground proposals caused by the varying intensity levels within the same sequence. Finally, they use a Deep Convolutional Neural Network (DCNN) with a configurable input size and convolutional layers to classify the proposed regions.

To exploit the sequence information present in the bursts, Jeff Donahue et al. [4] used an LSTM layer by wrapping each image in a sequence in separate convolutional layers. Each image in the sequence would be wrapped inside a TimeDistributed layer, where each image would be treated as one time stamp. Once the features were generated for each image inside a sequence, an LSTM layer would then be added after this to see if any information can be extracted from the features of the images.

Jeff Donahue et al. [4] used a new method such that instead of taking the output of only the last cell from the LSTM layer, they took the output of each cell and average the predictions to get one final prediction. This was incorporated in our research.

Jhony Heriberto et al. [8] proposed a multi-layer robust principal component analysis (multi-layer RPCA) approach for background subtraction. Their method computes sparse and low-rank images from a weighted sum of descriptors, using color and texture features as case of study for camera-trap images segmentation. The segmentation algorithm is composed of histogram equalization or Gaussian filtering as pre-processing, and morphological filters with active contour as post-processing. They optimize the parameters of their multi-layer RPCA using an exhaustive search.

Marco Willi et al. [23] used Convolutional neural networks (CNNs) in their work to differentiate among images of different animal species, images of humans or vehicles, and empty images. They worked with a combination of many famous datasets and showed that with transfer learning, the performance improvement in classification is quite significant in this domain. Our interesting approach to use transfer learning to initialize non-standard networks (which vary in the size of the input layer) was motivated by their gain in performance.

5 Methodology

This section describes the data pre-processing steps, the training and inference pipeline software framework implementation details, followed by baseline model approaches and sequence-model approaches.

5.1 Data Processing

The Wellington dataset [1] is split into two zip files: one containing all the images and the other containing the metadata. The metadata is a CSV file which specifies which image belong to which sequence and the label associated with it amongst other details.

During our initial exploration, we found that not all sequences had 3 images. We had to discard such sequences for the purpose of maintaining uniformity. We also excluded the sequences which were 'Unclassifiable' since we could not say what the true labels were. We binarized the label to whether an animal is present or not. Finally, we created a new tabular dataset by pivoting the existing one so that each new row represents a sequences and contains references to the images that belong to this sequence and the label associated with them. We opted for this structure in order to simplify the ingestion of data in the pipeline downstream. We then split this dataset into train, validation and test set in the ratio 0.7, 0.15 and 0.15 respectively, by randomly sampling. There were 62165 sequences for training, out of which 51426 sequences contained an animal while 10738 sequences were empty (did not contains any animal). We up-sampled the empty-sequences to balance the class distribution. However, we perform down-sampling (random removal) of sequences with animals in order to balance the validation and test sets, which resulted in 4279 and 4601 sequences in validation and test sets respectively. Also, all images were resized to 224 pixel-square in order to work with a manageable size for running the computations in a reasonable amount time.

Finally, we built a highly-configurable data pipeline using the Dataset API provided by TensorFlow [15] that would work with sequences of any sizes. This pipeline handles loading, processing, batching and pre-fetching of images from the disk in order to speed up the training process. The pipeline performs random augmentations such as flipping, color setting, etc. on the sequence level. Since the input data is a sequence of three images, the output required from the pipeline varies for the various models that we plan to work with. Some require only one of the images from the sequence, some require all three as separate data points and some require a single image with 4 channels. The pipeline allows for all this flexibility in by setting the 'mode' configuration while creating the object.

5.1.1 Challenges

- Pre-processing cost: Working with gradient descent based optimizers entails making multiple passes over the dataset. Since our dataset is 190 GB on the disk, reading, holding in memory and processing this data takes up a lot of memory and time. In order to tackle this limitation, we need to reduce on-the-fly operations as much as we can. We do this by first performing the resizing operation before the training process begins. We pick the size of the image based on the input layer requirement of the pre-trained network weights that we plan to use. Then, we batch the dataset and perform random augmentations and other processing on the fly. Batching helps reduce the memory load on the GPU. If a mode requires a single image from a sequence, we avoid reading the other images of the sequences in order to speed up the training process.

- Disk bottleneck: As explained in the previous process, reading, augmenting and generating data points is an expensive operations. The actual gradient calculation is, however, relatively faster making the loading operation a bottle neck. We tackle this issue by pre-fetching multiple subsequent in parallel batches and keeping them in memory for the GPU to use when it finishes processing the current batch.

5.2 Model Training and Inference Framework Implementation

We use Azure Blob storage and Azure computes to store and run our models respectively. In order to quickly load the data onto different VMs, we have stored the data on an Azure Disk, which we can easily mount to different VMs. We would be using TensorFlow 2.0 for implementing our data pipelines and model architectures due to its popularity and well-documented user guides. All of our code would be in Python 3.

We designed and implemented a framework which has independent data pipeline, model architectures, training pipeline and inference (evaluation) pipeline. This allows us to work independently on different models, and easily integrate and test our solutions.

5.3 Baseline Models

As introduced in the section 3, we consider single-image based models (which do not utilize any form of sequence information) as baseline models. During training time, all images from a sequence are considered as independent data-points having the same label as the one associated with the sequence, and therefore, all images from all training sequences are used independently to train the baseline models. However, during inference time, only the first-image of each sequence in the test-set are used to obtain the predicted labels, in order to maximize the chance of an animal being present in the frame since this image is captured as soon as the sensor on the camera gets triggered.

We experimented with various standard CNN architectures like VGG-16 [13], InceptionResNet-V2 [14], and variations of ResNet [9] like ResNet-50 [19], ResNet-101 [16], ResNet-152 [17], and ResNet-152-V2 [18]. These architectures were chosen as they have been proven to perform well on ImageNet [3] challenge. We initialized the network with pre-trained weights from ImageNet prior to training to achieve faster and better convergence.

5.4 Proposed Sequence Models

As described in section 6, ResNet-152 architecture performs well as a baseline (single-image classifier) compared to other network architectures. Therefore, ResNet-152 architecture is used as the backbone feature-extractor for the sequence-models.

We experiment with the following 2 subsection describe the proposed approaches for sequence models. The figure 2 shows a pictorial overview of our approaches.

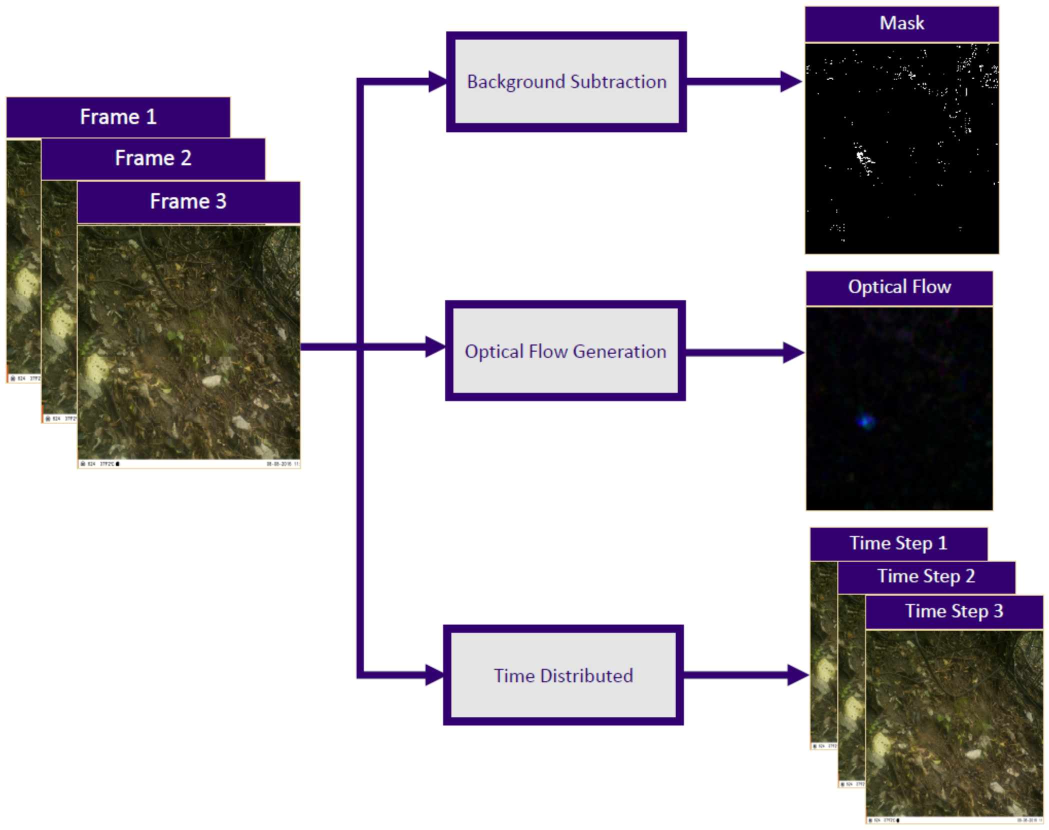

Figure 2: Extraction of sequence information from the 3 images in the sequence.

5.4.1 Recurrent Neural Architecture

LSTM [10] network architecture is proven to work well in learning sequence information, especially in Natural Language Processing domain. Therefore, we considered this method to learning the sequence information present in our short-sequence of images. The feature maps from ResNet-152 for each of three images in each sequence passed through LSTM cells. This model is referred as 'LSTM' model in the rest of the paper.

In this model, each frame is considered as a time step for the LSTM layer. A TimeDistributed wrapper is used over each image in the sequence, such that the base CNN-architecture is used for each image in the sequence. Thus, each image in a sequence passes through a CNN, to produce features. The features of all the images in the sequence are then considered as time stamps and passed onto the LSTM layer. Note that for LSTM model, the backbone architecture used was ResNet-50 instead of ResNet-152.

5.4.2 Sequence Information Channel Concatenation

The sequence information is extracted as a mask from the three images of the burst using either of the following two methods. The below methods were considered due to their proven effectiveness in motion analysis research. The resulting masks are concatenated to the first-image or all three images of the sequence across channels and is considered as the input-layer of the ResNet-152 model.

- **MOG2 Mask**: This is an OpenCV implementation of a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It is based on two papers by Z.Zivkovic [25][26]. One important feature of this algorithm is that it selects the appropriate number of Gaussian distributions for each pixel and hence, it provides better adaptability to varying scenes due to illumination changes etc. We create a background subtractor object by passing all the frames of the sequence to the function call. This then returns us a grayscale mask which differentiates between the background, foreground and the shadows.
- **Optical Flow RGB Mask**: Dense Optical Flow using Gunnar Faurneback's algorithm [5] results in an RGB mask (colors represent direction of motion). This method takes 2 images in a sequence and produces a mask. Since we have 3 images in our sequence, we get two Optical Flow masks - Optical Flow RGB Mask-1-2 computed between first and second image, and Optical Flow RGB Mask-2-3 computed between second and third image. However, for most of our experiments we take a simple average of these two masks which is referred in the rest of the paper as Optical Flow RGB Mask.

The comparison of the masks produced by MOG2 and Dense Optical Flow for few sequences have been shown in Figure 3, which provides insight on how the sequence information is being highlighted.

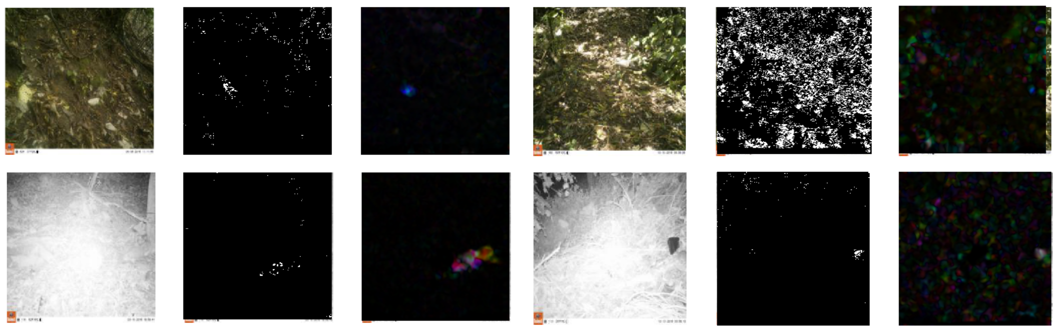

Figure 3: Comparison of MOG2 grayscale mask and Optical Flow RGB mask. The figure shows the first image in the sequence, MOG2 mask and Optical Flow RGB mask for the 4 sequences shown in Figure 1.

We experiment with a variety of channel-wise concatenation possibilities, by modifying the first (input) layer of the model. Note that we initialize the channels corresponding to original image (either first image or all the images from the sequence), and the channels corresponding to the optical-flow output (RGB channels) with the ImageNet pre-trained weights, but use a random-normal initialization for the MOG2 grayscale mask. The following are the variations of the Sequence Information Channel Concatenation approach:

- **MOG2-4-Channel**: Image1 (3) stacked with MOG2 greyscale mask (1) across channels to form 4-Channel input layer.
- **MOG2-10-Channel**: Image1 (3), Image2 (3) and Image2 (3) stacked across channels along with MOG2 greyscale mask (1) to form 10-Channel input layer.
- **OpticalFlow-6-Channel**: Image1 (3) stacked with Optical Flow RGB mask (3) across channels to form 6-Channel input layer.
- **OpticalFlow-15-Channel**: Image1 (3), Image2 (3) and Image2 (3) stacked across channels along with Optical Flow RGB mask-1-2 (3) and Optical Flow RGB mask-2-3 (3) to form 15-Channel input layer.
- **Hybrid-13-Channel**: Image1 (3), Image2 (3) and Image2 (3) stacked across channels along with Optical Flow RGB mask-1-2 (3) and MOG2 greyscale mask (1) to form 13-Channel input layer.

- **OpticalFlow-Only (6-Channel)**: Optical Flow RGB mask-1-2 (3) and Optical Flow RGB mask-2-3 (3) stacked across channels to form 6-Channel input layer.
- **OpticalFlow-MOG2-only (7-Channel)**: Optical Flow RGB mask-1-2 (3), Optical Flow RGB mask-2-3 (3) and MOG2 greyscale mask (1) stacked across channels to form 7-Channel input layer.

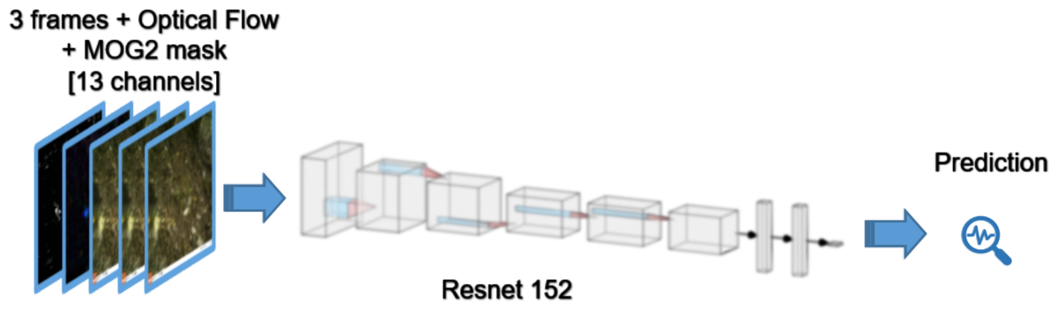

Figure 4: Concatenated Input Layer along with the backbone CNN architecture shown for the Hybrid-13-Channel.

Image1, Image2 and Image2 refer to the first, second and third images of the sequence respectively. Figure 4 shows this approach pictorial for the Hybrid-13-Channel model described above.

Each of the above configuration is evaluated on the held-out test-set, and the ROC AUC is compared across baseline and difference sequence-models, to determine if the sequence information can be effectively used to improve the classification performance.

6 Results and Discussion

In this section, we describe the selection of the best performing baseline-model, which would henceforth be used as a backbone CNN architecture for the modeling approaches described in section 5. Further, we compare the performance of variations of sequence-models to the baseline model, followed by the discussion to determine whether the sequence-information is helpful to our problem.

The comparison of the performance of these baseline-model architectures on the test-set has been summarized in this Table 1. We observe that the ResNet-50 and ResNet-152 architecture provide the highest ROC AUC of 0.85. Therefore, we consider ResNet-152 as the backbone CNN architecture for sequence-models (as more parameters than ResNet-50 maybe helpful for sequence-models, and this also implies that the ResNet-152 architecture is a good feature extractor from single-images). If any of the sequence models can achieve a better performance on the test-set, then it would imply that sequence information is useful for our problem.

The Table 2 shows the performance of all the sequence-models compared to the baseline model on the test-set. We observe that the best-performing sequence model is **"hybrid-13-channel-model"** which provides about 11% increase in ROC AUC utilizing the same ResNet-152 backbone feature extraction layers. Therefore, we claim that the sequence information captured in MOG2 and Dense Optical Flow outputs are effectively utilized by concatenating across the channels in the input-layer, to achieve a significantly better classification of 3-image-sequence instances of camera-trap data.

From Table 2, we observe that the Optical Flow RGB masks independently provide better performance than the baseline models, with MOG2 inclusion giving a slight improvement along with Optical Flow RGB masks. However, these masks when used along with the original image (by concatenating across channels with either first-image or all three images of the sequence) is found to be much more effective. The LSTM model might not be performing well as it maybe effective for sequences of longer length which are usually found in Natural Language Processing domain.

Model Architecture	ROC AUC
VGG-16	0.6
ResNet-50	0.85
ResNet-101	0.84
ResNet-152	0.85
ResNet-152-V2	0.84
InceptionResNet-V2	0.84

Table 1: Performance comparison of single-image based Baseline-Models on held-out test set. We observe that ResNet models perform well, with ResNet-50 and ResNet-152 providing the best performance. However, considering better flexibility, ResNet-152 is considered as the backbone architecture for sequence-based models.

Model Name	Model Description	ROC AUC
Baseline	Single-image based ResNet-152	0.85
LSTM	LSTM cells over Baseline feature maps	0.88
MOG2-4-Channel	Image1 + MOG2 mask	0.93
MOG2-10-Channel	Image1 + Image2 + Image3 + MOG2 mask	0.95
OpticalFlow-6-Channel	Image1 + Optical Flow RGB Mask	0.96
OpticalFlow-15-Channel	Image1 + Image2+ Image3 + Optical Flow RGB Mask	0.95
Hybrid-13-Channel	Image1 + Image2 + Image3 + Optical Flow RGB mask + MOG2 mask	0.96
OpticalFlow-Only (6-Channel)	Optical Flow RGB Mask-1-2 + Optical Flow RGB Mask-2-3	0.90
OpticalFlow-MOG2-only (7-Channel)	Optical Flow RGB Mask-1-2 + Optical Flow RGB Mask-2-3 + MOG2 mask	0.91

Table 2: Comparison of all the model performances on the held-out test set. We find that sequence-models perform significantly better over the Baseline model, with the OpticalFlow-6-Channel model and Hybrid-13-Channel model giving the best performance.

7 Conclusion and Future Work

We observe that sequence-information concatenated across channels with the input image provides a significant improvement of 11% in the ROC AUC over baseline single-image model with the same base CNN architecture. Both MOG2 mask and Optical Flow mask seem to help, but Optical Flow mask performs slightly better than MOG2 mask. The **OpticalFlow-6-channel model** listed in table 2 can be considered as the best sequence-model as it provides the best ROC AUC with the least number of input channels. We demonstrate that the sequence information can be exploited to improve the classification in camera-trap images, as the animals may be hard to spot in individual images, but are visible when observed in the short-burst of images. Therefore, camera trap deployments can consider taking bursts of images instead of single images.

Future work can expand this work to test whether similar improvement can be observed in a multi-class species classification setting. Future work can also consider determining the optimal length of the sequence, by experimenting with sequences of different length, which might be helpful for appropriate camera trap deployments. It could be expanded to observe how such channel-wise concatenation helps in detector-based setting where models utilize strong-labels in terms of bounding box or segmentation information.

References

[1] V. Anton, S. Hartley, A. Geldenhuis, and H. U. Wittmer. Monitoring the mammalian fauna of urban areas using remote cameras and citizen science. *Journal of Urban Ecology*, 4(1):juy002, 2018.

[2] L. BC. Labeled information library of alexandria: Biology and conservation, 2019.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[5] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[6] M. A. for Earth. Cameratraps, 2019.

[7] M. A. for Earth. Speciesclassification, 2019.

[8] J.-H. Giraldo-Zuluaga, A. Salazar, A. Gomez, and A. Diaz-Pulido. Camera-trap images segmentation using multi-layer robust principal component analysis. *The Visual Computer*, 35(3):335–347, 2019.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] Microsoft. Coco camera traps format, 2019.

[12] Microsoft. Microsoft ai for earth, 2019.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[15] TensorFlow. A collection of datasets ready to use with tensorflow., 2019.

[16] TensorFlow. Tensorflow keras applications - resnet101., 2020.

[17] TensorFlow. Tensorflow keras applications - resnet152., 2020.

[18] TensorFlow. Tensorflow keras applications - resnet152v2., 2020.

[19] TensorFlow. Tensorflow keras applications - resnet50., 2020.

[20] W. Wei, G. Luo, J. Ran, and J. Li. Zilong: A tool to identify empty images in camera-trap data. *Ecological Informatics*, 55:101021, 2020.

[21] Wikipedia contributors. Camera trap, 2019. [Online; accessed 20-Nov-2019].

[22] Wikipedia contributors. Receiver operating characteristic, 2019. [Online; accessed 12-Dec-2019].

[23] M. Willi, R. T. Pitman, A. W. Cardoso, C. Locke, A. Swanson, A. Boyer, M. Veldthuis, and L. Fortson. Identifying animal species in camera trap images using deep learning and citizen science. *Methods in Ecology and Evolution*, 10(1):80–91, 2019.

[24] H. Yousif, J. Yuan, R. Kays, and Z. He. Animal scanner: Software for classifying humans, animals, and empty frames in camera trap images. *Ecology and Evolution*, 9(4):1578–1589, 2019.

[25] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 28–31. IEEE, 2004.

[26] Z. Zivkovic and F. Van Der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.