CS230

Counting Animals in Photo-Sequences using Deep Learning

Mayur Deshpande [*]
SCPD, Stanford University
nep@stanford.edu

Abstract

I present two deep learning methods to estimate animal counts from a sequence of photos (trap-camera burst shoots) via: (a) An LSTM network that learns from bounding boxes, (b) a 3D convolutional network that learns from segmentation masks. While MOTS (Multiple Object Tracking Systems) exist for human and vehicle counting in videos, they typically employ algorithms such as Kalman filtering for final count estimation. In this paper, I present methods to directly deep-learn the final count. The LSTM network performs better than the 3D-conv network, achieving less than 1.0 mean error on count from ground truth.

1 Introduction

Estimation of wildlife population is manual and costly. Recently, instead of manual sighting, wildlife conservationists have been deploying camera traps that take multiple quick photos when triggered by animal movements. The photo(sequences) are then analyzed for wildlife population estimation [7]. However, even this can be time-intensive and costly. Automating animal counts from a burst-sequence can greatly help cost and time.

A photo-burst increase the likelihood of capturing the animal(s) of interest. The intervals between the frames in a sequence can vary depending on the camera used, typically from 0.1 to 1 second. A photo burst typically consists of 10 frames, so lasting anywhere from 1sec to 10 seconds per sequence.

1.1 Challenges

Precise counting across the sequence is challenging. Consider the example of a herd of animals passing by the camera. Frame-1 may record 2 animals, frame-2 may record 3 (2 of them from frame-1), frame-3 and frame-4 are empty and frame-5 may record 2 animals (new animals). The total in this sequence is 5 animals.

The problem to be solved is then this: (a) The input is a sequence of images (jpeg/RGB) each labeled with an id that belong to one sequence (sequence_id). (b) The output is a number (integer) indicating the total number of unique animals detected in the sequence.

A naive approach would be to run object(animal) detection per frame and sum all animals detected. However, this will typically lead to over-counting. In example above, this would give us 7 animals. Conversely, another naive approach might be to take the max of objects detected in a sequence. But this would lead to an undercount: 3 animals in example above. Thus, we need a way to track animals across somewhat sparse temporal samples.

[*]Thanks to Jack Lee (jacklyonlee@stanford.edu) for his valuable insight to try an LSTM approach.

There are three further challenges:

1. **Occlusion** A animal can be completely hidden by another animal is a particular frame[2]
2. **Exit** If an animal is detected as missing from one frame to next, did it actually exit the frame or did it get occluded?
3. **Entry** If a 'new' animal is detected in a frame, is it actually new or is it the same animal that exited the frame and has re-entered?

In this paper I evaluate two approaches where the deep-learning network can hopefully track animals correctly across the frames and directly output the estimated animal count.

1. **LSTM Approach**
   The images the images are first pre-processed to generate bounding-boxes: one bounding box per animal detected plus a confidence score for the detection. Each frame in a sequence corresponds to one time-step to the LSTM network. The input in each time-step is set of bounding-boxes(plus confidences) for that frame.
2. **3D Convolution Approach**
   The images the images are first pre-processed to generate segmentation-masks as .png images (height x width x 1 channel). The network treats all the frames in a sequence as a single volume to convolve over.

2 Related work

In[7], the authors use deep learning to identify animal species present in camera-bursts. However, they leave the work of accurately counting the total number of animals to future research.

MegaDetector[4] identifies if animals are present in a single frame and can generate two valuable outputs: (a) Segmentation mask boundaries around the animals detected (using DeepMac[3]) and (b) Bounding box co-ordinates for the animals detected. I rely on MegaDetector and DeepMac to generate the outputs from the raw images. However, MegaDetector cannot estimate the total number of animals from a sequence of frames.

In research outside of the animal context[2,8], the problem is typically split into two phases. In the first phase, the bounding boxes for each object of interest are detected. Then a set of algorithms are used to keep 'track' of objects across frames: (a) A distance-calculation algorithm first calculates how much a bounding-box moved across two frames and (b) A Kalman-filter is applied to calculate the velocity and position across frames and finally (c) An assigment algorithm, like the Hungarian method[6], is used to assign bounding-boxes to objects. This has shown promising results when applied to pedestrian and vehicle counting in video data. No research exists as far as I know that does this for wildlife context. Further, it is an open research question whether this can be done in "one-shot", i.e., whether the neural network can directly output the final animal count.

3 Dataset and Features

The data is already provided as part of the Kaggle iWildCam 2022 competition[9]. The dataset (among other things) provides count annotations on 1780 sequences. Each sequence can contain anywhere from 1-10 frames per camera. Most locations have only one camera but some locations can have two cameras, giving up to 20 frames per camera. Different cameras output different resolution images, from 720x1280 upto 2226x3565 pixels (RGB).

To keep things simple to start with, I filtered out sequences with more than one camera. The final Train/Dev/Test split was: **Train:** 1009 sequences, **Dev:** 216 sequences, **Test:** 216 sequences

1. **LSTM pre-processing:** For the LSTM network, I used the DeepMac generated bounding boxes. Each sequence was standardized to 10 frames (empty frames contained all zeros). I made a pass through all frames and saw that maximum detections was 23, so I standardized to 25 detections per frame. Each detection is a 5-tuple: (confidence, x-top, y-top, x-bottom, y-bottom). This resulted in a data-set of shape: (1009, 10, 25, 5). To be conducive for LSTM, the last two dimensions were collapsed, so each frame can be thought of as a

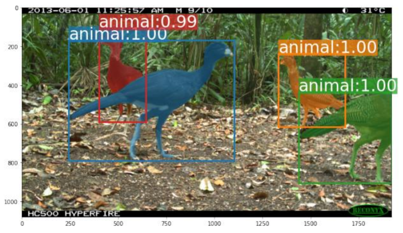

Figure 1: Segmentation-Masks and Bounding-Boxes

125 dimensional vector. The input to LSTM for one input was then: (10, 125). Only animal-detection bounding boxes were used (vehicle and human detections were filtered out).

2. **3D-Conv pre-processing**: For the 3D-Conv network I used the DeepMac generated segmentation masks which retains the original image's height and width dimensions but is single channel png image with 4 values (0: background, 1: animal, 2: person, 3: vehicle). I standardized all segmentation mask images to 1024x1280 via image.resize() using LANCZOS algorithm. This resulted in a data-set of shape: (1009, 10, 1024, 1280, 1).

In Figure-1 we can see a high-resolution color image with segmentation masks and bounding box detections applied to it.

4 Methods

4.1 Loss and Accuracy

Output For both the approaches, the final output layer is one unit with ReLU activation. This was chosen since ReLU is a (rectified) linear unit and it's output space is the real-number line.

Loss Function We want the network to get the predicted number of animals as close to the labeled number as possible. Either undershooting or overshooting is bad, so absolute error makes sense. Further, most labeled sequences are small numbers (less than 15) so we want the loss to be high when predictions are not close to ground-truth. Putting these together, I chose mean exponential loss.

$$Exponential\_Loss = \frac{1}{n}\sum 2^{(y\_pred - y)}$$

I also tried with a more traditional (square) loss function:

$$Squared\_Loss = \frac{1}{n}\sum (y\_pred - y)^2$$

Prediction Error For the prediction error, I used a metric that Kaggle would use for evaluation of the model: as simple Mean Absolute Error:

$$Error = \frac{1}{n}\sum |y\_pred - y|$$

4.2 The Models

For both the models described below, I used the ADAM (adaptive moments) gradient descent optimization algorithm.

1. **3D-Convolution:** The basic idea here is to feed segmentation-mask frames of a sequence as one volume to the network and have it output the predicted number. I used two 3D-convolution layers. The full pipeline is: Input->Conv3D->MaxPool->Conv3D->MaxPool->FC(ReLU)->FC(ReLU). Parameters for each stage are given in the appendix. The pattern followed is of a typical Conv-net with width and height reducing at each stage and number

of filters increasing. Since the training examples are limited (1000), I wanted to keep the total trainable parameters low. The hope here is that the network will automatically learn an capture uniqueness of bounding-boxes across the full sequence when it makes a convolution over the entire volume.

2. **LSTM:** I used a double layer LSTM network (similar to the class 'Emojify' assignment): the first LSTM layer is bidirectional and the outputs are sent to a forward-LSTM. The output of the second LSTM is fed into fully-connected ReLU unit. The full pipeline is: Input->LSTM(bi-directional)->Dropout->LSTM(forward)->Dropout->FC(ReLU)–>FC(ReLU). The hope is that the network will automatically learn the overlap in bounding boxes across the sequence while still being able to 'remember' and give appropriate weightage to bounding-boxes that only appeared once or twice in early part of sequence (animal exited early).

I developed all code using Tensorflow[1] and Keras[5] frameworks, trained using TPU.

5 Experiments/Results/Discussion

Model	Hyerparameters	Train-Loss	Train-Error	Test-Loss	Test-Error
Baseline-(0)	NA	NA	NA	NA	3.5
Baseline-(3)	NA	NA	NA	NA	1.78
Conv3D exp_err	Batch=16, Epochs=80	4.3	1.8	5.3	2.0
LSTM-2 sq_err	Batch=16, Epochs=120	0.46	0.48	1.9	0.93
LSTM-2 sq_err	Batch=16, Epochs=80	0.8	0.6	1.9	0.95
LSTM-2 exp_err	Batch=16, Epochs=80	1.8	0.7	2.5	0.88
LSTM-1 sq_err	Batch=16, Epochs=80	0.98	0.69	1.8	0.93
LSTM-1 exp_err	Batch=16, Epochs=80	2.1	0.77	3.4	0.96

5.1 Quantitative Results

1. **Baseline:** To establish a baseline, we measure the predicted error on test set when prediction is always set to a particular number. So Baseline-0 implies prediction is always 0. As table above shows, Baseline-0 had an error of 3.5 – this shows that the average number of animals in a sequence is quite low (around 3-4). The maximum in the test-set is 9. Further, I evaluated different baselines and Baseline-3 has the lowest error of 1.78. So, any model we develop should hopefully perform better than that.

2. **3D-Convolution:** This model did quite poorly. The performance of Conv3D using the exponential-loss function, a batch size of 4 is shown in table above. The test-error is 2.001, which is even higher than baseline-3. I tried some simple variations (batch size, loss-functions) but above was the best result I got.

3. **LSTM:** LSTM-2 indicates a model with 2 LSTM layers. LSTM-1 indicates a model with only one LSTM layer – the base bidirectional layer has been removed from LSTM-2. From table above, we can see that LSTM-2 trained with exponential loss function performs best, achieving an error of 0.88. This indicates that the network is off on the count on average by 1. The variance is 1.05 which is also quite low. Training epochs of 80 seemed to yield the best results. Training for longer (120 epochs) reduced the training-error but led to overfitting and performing slightly worse on the dev set. Using the exponential loss function gave slightly better results than using squared loss. LSTM-1 though slightly worse than LSTM-2 performed much better than Conv3D achieving a best case error of 0.93.

5.2 Qualitative Analysis

For the LSTM model, it seems like it does poorly on night-image captures and those with many entries/exits. Example of one such image is presented in Fig-2.

As a non-expert, we can see that there are 3 animals in the first five frames (frames go top:left to right, then bottom: left to right). In the seventh frame a new animal appears and in the 10 frame there seems to be yet another new animal.

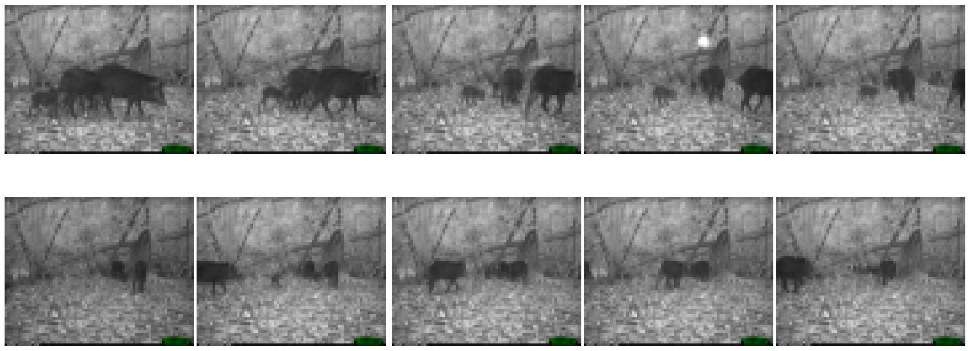

Figure 2: A Tough Sequence: 3, 4, or 5 unique animals?

In this particular sequence, my guess was 4 or 5. The network predicted 4 but the actual ground truth by an expert is 9!

5.3 Discussion

If I had more time, I would most likely tried the following two things.

1. In general, the training data set was quite limited (1000) so if there was more time I would have tried to augment the data set. One idea for the LSTM would be to take the bounding boxes and slightly perturb them (move them slightly horizontally or vertically) randomly.

2. The LSTM model currently works on just the bounding boxes but that leaves out many distinguishing features of the animal in the bounding box. It might be really interesting to try a neural transfer approach, taking the DeepMac model and freezing most of it's initial layers. Then maybe take the last 1-2 layers and feed those directly to the LSTM.

6 Conclusion/Future Work

In this paper, I developed and evaluated two deep-learning neural network models to predicted total animal sightings from a burst-sequence of trap-cameras. The work built upon impressive technologies and models available today that can classify a wide range of animal species and accurately generate bounding boxes and segmentation masks from raw images.

The Convolutional model performed poorly while the LSTM model was able to predict within an error range of +/- 1.0. This is exciting since we can now try to do one-shot count estimation from raw frames without using any kind of post-processing algorithms. Further, the error of the 1-layer LSTM was not that much worse than the 2-layer LSTM. Further, the 1-layer LSTM is uni-directional 9and quite small) suggesting it could be used in a resource constrained and online scenario as well.

Though the error rate seems tolerable, it would be nice to bring it down even further. Directly integrating the LSTM into the DeepMac model might be one way to do this; and probably in combination with more sophisticated loss functions that push the network to reduce it's error rate.

References

[1] Abadi, Martín & others (2015) TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/

[2] Bewley, A., Ge, Z., Ott L., Ramos F., Upcroft B. (2017) Simple Online and Realtime Tracking, *arXiv:1602.00763*

[3] Birodkar V., Lu, Z., Li, S., Rathod, V., Huang, J. (2021) The surprising impact of mask-head architecture on novel class segmentation *arXiv:2104.00613*

[4] Beery S., Morris, D., Yang, S. (2019) Efficient Pipeline for Camera Trap Image Review *arXiv:1907.06772*

[5] Chollet, François & others (2015) Keras. https://keras.io

[6] Kuhn, H. W. (1955) The Hungarian Method For The Assignment Problem, *Naval Research Logistics Quarterly*, pp. 83-97

[7] Norouzzadeh, M. S., Nguyen, A., Kosmala, M. & others (2018) Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *PNAS https://doi.org/10.1073/pnas.171936711*

[8] Wojke, N., Bewlwy, A., Paulus, D. (2017) Simple Online and Realtime Tracking with a Deep Association Metric, *arXiv:1703.07402*

[9] (2022) Kaggle iWildCam 2022 Competition *https://www.kaggle.com/competitions/iwildcam2022-fgvc9/overview*

7 Appendix

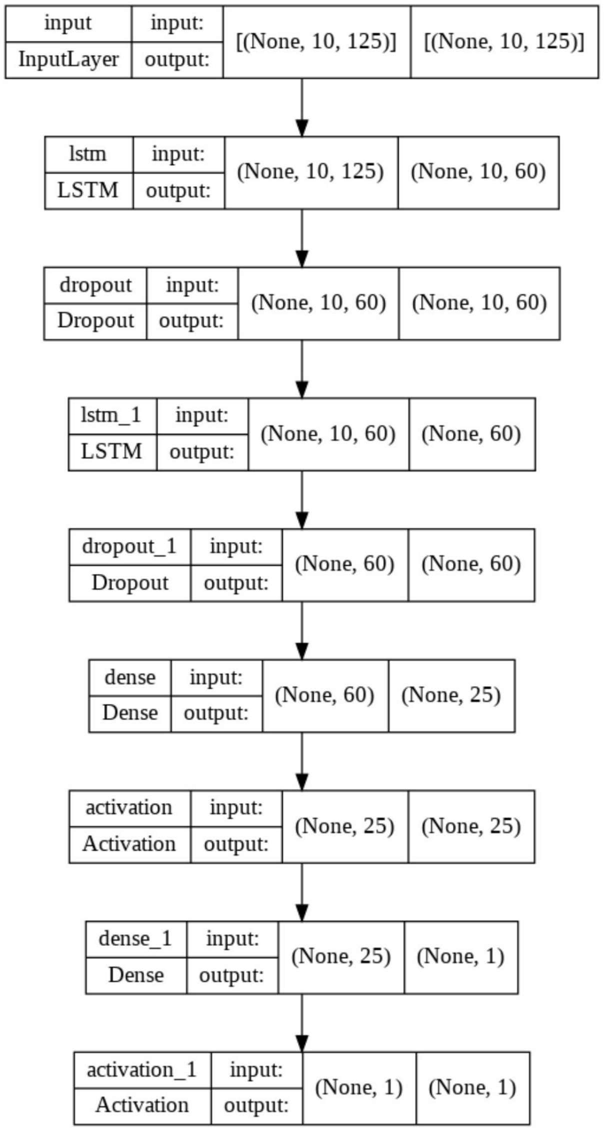
input | InputLayer | input: [(None, 10, 125)] | output: [(None, 10, 125)]
lstm | LSTM | input: (None, 10, 125) | output: (None, 10, 60)
dropout | Dropout | input: (None, 10, 60) | output: (None, 10, 60)
lstm_1 | LSTM | input: (None, 10, 60) | output: (None, 60)
dropout_1 | Dropout | input: (None, 60) | output: (None, 60)
dense | Dense | input: (None, 60) | output: (None, 25)
activation | Activation | input: (None, 25) | output: (None, 25)
dense_1 | Dense | input: (None, 25) | output: (None, 1)
activation_1 | Activation | input: (None, 1) | output: (None, 1)

Figure 3: LSTM Model

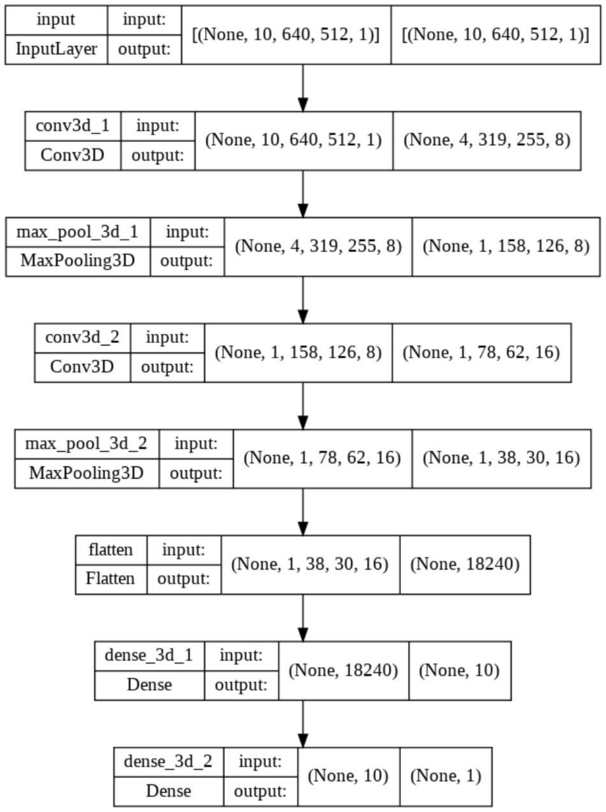
input | input: | [(None, 10, 640, 512, 1)] | [(None, 10, 640, 512, 1)]
InputLayer | output:
conv3d_1 | input: | (None, 10, 640, 512, 1) | (None, 4, 319, 255, 8)
Conv3D | output:
max_pool_3d_1 | input: | (None, 4, 319, 255, 8) | (None, 1, 158, 126, 8)
MaxPooling3D | output:
conv3d_2 | input: | (None, 1, 158, 126, 8) | (None, 1, 78, 62, 16)
Conv3D | output:
max_pool_3d_2 | input: | (None, 1, 78, 62, 16) | (None, 1, 38, 30, 16)
MaxPooling3D | output:
flatten | input: | (None, 1, 38, 30, 16) | (None, 18240)
Flatten | output:
dense_3d_1 | input: | (None, 18240) | (None, 10)
Dense | output:
dense_3d_2 | input: | (None, 10) | (None, 1)
Dense | output:

Figure 4: 3D-conv Model