1. Prerequisites and Dependencies

1.1. **Technical dependencies.** The software requires

- Python ≥ 3.7
- Tensorflow with GPU support ≥ 1.14 (tested on 2.1), for training the object detector, Tensorflow 1.14 is required.
- Keras ≥ 2.3.1 (tested on 2.3.1)
- the efficientnet keras implementation [4]
- The matterport mask rcnn implementation [3]
- various libraries and packages
  - openpyxl
  - imgaug [2]
  - opencv (tested on 4.4.0.44) [1]
  - matplotlib
  - sklearn
  - shutil
  - numpy
  - csv
  - copy
  - datetime
  - pickle

1.2. **Prerequisites on the data for inference.** We expect that the user wants to classify standing, lying-head up and lying-head down as well as being absent. The program code is easily extendable to various postures.

The data has to be given as .avi video files with a frame-rate of one frame-per-second. Furthermore, we expect any individual to be uniquely determined by the *species name*, *zoo name* and *individual number*. Species name and zoo name may not contain special characters while the individual number is an integer ≥ 1.

We define one *enclosure* as the box where multiple individuals of one species may be contained in. Furthermore, an enclosure can be filmed by multiple cameras.

Thus, for each species in each zoo, there is a unique *enclosure number* (which the individual belongs to) and there are *video numbers* corresponding to the different cameras used. Given a zoo name, each video number belongs to exactly one enclosure.

Example. In the EXAMPLE-ZOO we are observing three common elands, one is stalled solely and recorded by camera 1. The second and third are stalled together and recorded by in total three cameras: 2 – 4. Thus,

species	zoo name	enclosure number	video number	individual number
CommonEland	EXAMPLE-ZOO	1	1	1
CommonEland	EXAMPLE-ZOO	2	2;3;4	2;3

Table 1. Example for correct usage of the keys.

The video files have now to be stored as follows:
.../SpeciesName/ZooName/SpeciesName_VideoNumber/YYYY-MM-DD_SpeciesName_ZooName_VideoNumber.avi

2. Training

2.1. **Object Detector Mask R-CNN.** Prepare your data in Pacal VOC style, thus one folder with images in jpg format and one folder of annotations as xml files.

Edit *training/training_mrcnn.py* by updating the variables of Table 2.

By running the script you will train all object detectors signified above.

2.2. **Behaviour Classifier EfficientNet.** Prepare your training data as common in classification tasks, thus create a folder containing a subfolder for each class with images. Take care that the classes are approximately balanced or edit the program code to use class weights during training.

Then just edit *training/training_efficientnet.py* by updating the variables of Table 3. By running the script you will train the behaviour classifier. The input images can be both, either images from single frames or the described multiple frame encoding in the paper.

variable	type	description
BASE_MODEL_PATH	string	Path to a base model. Can either be the pretrained network on COCO provided by Waleed [3] or a previously trained object detector.
IMAGES_PATHS	list of lists of strings.	Each string signifies a folder containing .jpg images. Strings in one list will be used jointly for training one network. Different lists can be used to train multiple networks on different sets subseqeuntly.
LABEL_PATHS	list of list of strings	The corresponding folders containing the annotation files.
AUGMENTATION	list of booleans	For each training sequence (see above), the boolean signifies whether image augmentation is applied.
NAMES_LABELS	list of list of strings	For each training sequence, the list contains all possible labels of the annotation files. If a label is missing, an error will occur during training.
NEW_MODEL_NAMES	list of strings	Contains the save path (ending with $.h5$ for each trained model.
NEW_MODEL_OUTPUT	list of strings	Contains the save path for the checkpoints created by training each model.
TRAININGSSTEPS_PER_EPOCH	list of integers	Signifies the number of training steps per epoch in each training sequence.
NUM_EPOCHS	integer	Signifies the number of epochs every training sequence is using.
LAYERS	string	Has to be set to *heads* or *all*. Signifies whether all layers (in each training sequence) are trained or if certain weights stay frozen. A detailed description is given by Waleed [3].
CONFIG_NAME	string	Name of the configuration.

TABLE 2. Variables which should be modified in the training script.

3. INFERENCE

3.1. **Requirements.** Make sure that your setup satisfies all technical dependencies. Download the code from this repository and create a folder containing all files. Furthermore, install the matterport implementation of mask-rcnn [3]. Edit MASK_RCNN_LIBRARY from *training - training_mrcnn.py* and *image_cutout_functions.py* according to the path.

3.2. **Preparation.** Inference works in three steps and we recommend using the code with an IDE like spyder. First, create a csv file signifying which videos will be predicted. Second, edit configuration.py and third edit and run predict_csv.py.

variable	type	description
GPU_USAGE	integer	Signifies which GPU in the system will be used. If you only use one GPU, set the value to 0.
NUM_GPUS	integer	Signifies the number of GPUs used for training. The current implementation only allows training on one GPU.
BS_PER_GPU	integer	Batch size for training. Depending on your image material and your graphic card, a value between 4 and 64 might be suitable.
NUM_EPOCHS	integer	The number of epochs the network is training.
SAVE_EVERY_EPOCH	integer	The script will save the current weights at the end of each x-th epoch as a .h5 model.
HEIGHT, WIDTH	integer	Image size, for the usage of EfficientNet B3 one should use 300.
DATA_PATH	string	Path to the training data.
VAL_PATH	string	Path to the validation data.
MODEL_SAVE_PATH_BASE	string	Path to the folder where the network and its checkpoints will be saved.
MODEL_NAME_BASE	string	Name of the .h5 file coming out of the training process. Has to end with .h5.

TABLE 3. Variables which should be modified in the training script.

3.2.1. *preparing a csv file.* We require a comma-separated csv file in which each line corresponds to one date from which the video should be predicted. Attention, one csv file may only contain exactly one combination of enclosure, individual and videos. More precisely, one line needs to read

YYYY-MM-DD,SPECIES,ZOO,ENCLOSURE_NUMBER,VIDEO_NUMBERS,INDIVIDUAL_NUMBER

where multiple video numbers have to be separated by ;.

3.2.2. *configuration.* Edit *configuration.py* by updating the variables of Table 4. Further variables can be adjusted if required but are either self-explanatory (like the video length or starting time) or commented within the code file (like the possibility to overlay some regions of the video or like adjusting the post-processing rules).

3.2.3. *prediction.* Edit *predict_csv.py* by updating GPU_TO_USE according to your configuration. The prediction pipeline is configured to use only one specific GPU, thus it is possible to run multiple instances on different GPUs in parallel.

Run the file and use predict_csv() in order to start inference. The tool will proceed as follows.

(1) Convert all videos to single images (4 per 7 seconds) and save them in the temporary storage.
(2) Use the mrcnn object detector to cut out the animal on these images and create single frame images as well as multiple-frame encoded images and save them in the temporary storage.
(3) Predict the behaviour based on the single frame images.
(4) Predict the behaviour based on the multiple-frame encoded images.
(5) Move the cut-out images from the temporary storage to the final storage.
(6) Merge the predictions and apply post-processing.

It is possible to skip certain steps (see the parameters of predict_csv()) if they were already done previously.

variable	type	description
INPUT_CSV_FILE	string	Input path to the csv file.
TMP_STORAGE_IMAGES, TMP_STORAGE_CUTOUT, FINAL_STORAGE_CUTOUT	string	Temporary and final path in which the images cutted out of the video as well as predicted by mrcnn will be saved.
FINAL_STORAGE_PREDICTION_FILES	string	Folder in which the AI system will save the output of the prediction pipeline. Furthermore, for each individual there will be an xlsx-file with certain statistical values of the video.
BASE_PATH_DATA	string	Base path in which above's structure of data begins. Has to end with a slash.
BEHAVIOR_NETWORK_JOINT, BEHAVIOR_SINGLE_FRAME	dictionary	Signifies the .h5 model file which will be taken for behaviour prediction. Searches for SPECIES_ZOO_INDIVIDUAL-NUMBER, if not found for SPECIES_ZOO and if not found either for SPECIES. Finally it outputs the basenet for antelopes. This can be changed in lines 137–165.
BASE_OD_NETWORK, ZOO_OD_NETWORK, ENCLOSURE_OD_NETWORK	dictionary	Analogously as before but for the object detector.
OD_NETWORK_LABELS	dictionary	For each object detection network we need to input the classes on which it was trained.

TABLE 4. Most important variables to adjust of configuration.py.

4. LICENSE

REFERENCES

[1] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[2] A. B. Jung. *imgaug*. `https://github.com/aleju/imgaug`. 2018.

[3] A. Waleed. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. `https://github.com/matterport/Mask_RCNN`. 2017.

[4] P. Yakubovskiy. *efficientnet*. https://github.com/qubvel/efficientnet. 2019.