Hainan Gibbon Classifier - User Manual

Introduction

In this study we describe the development of a classifier for identifying Hainan gibbon (Nomascus hainanus) calls in passive acoustic recordings collected as part of a long-term monitoring project whose conditions illustrate both the appeal and difficulty of automation. Hainan gibbons are one of the world's rarest mammal species, with fewer than 30 individuals believed to exist in the wild. Our goal was to develop an automated classifier for the monitoring project, together with software allowing the classifier to be run without deep learning or programming expertise. Below, we describe the usage of the software.

Authors

Emmanuel Dufourq, Ian Durbach, James Hansford, Sam Turvey, Amanda Hoepfner

Repository link

<https://github.com/emmanueldufourq/GibbonClassifier>

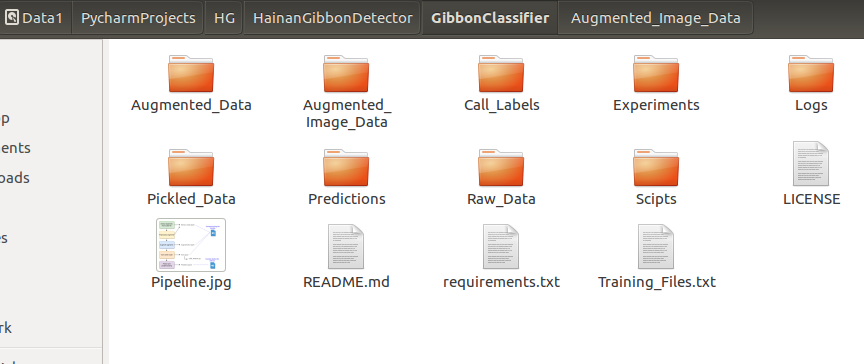
Requirements

*Install all requirements using pip install -r requirements.txt*

* Numpy
* Scipy
* Librosa
* Pandas
* Keras
* Sklearn
* Pickle
* Matplotlib

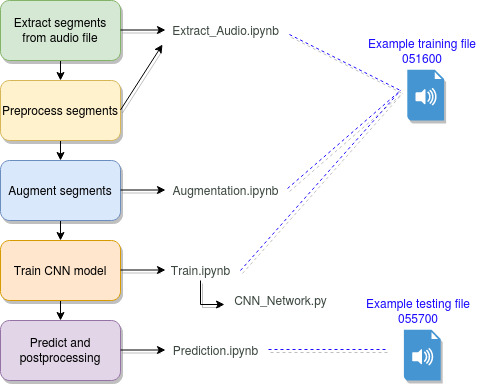
Initial folder structure

After cloning the github repository the folder structure of the project will be as shown below. The Python notebooks and scripts are in the “Scripts” folder.



Pipeline

Two files are provided to illustrate how to extract the audio segments, pre-process, augment the data and convert to spectrograms which allow the user to train the CNN. The testing file is provided to illustrate how to predict and obtain the output files. The provided training file is used in “Extract\_audio.ipynb”, “Augmentation.ipynb” and “Train.ipynb”. The testing file is only used in “Prediction.ipynb”. Both files are 8 hour long .wav files containing hainan gibbon calls. The training file has been annotated and the labelled files are in the “Call\_labels” folder.



Illustrating the overall pipeline for training and prediction.

Usage

Each step of the paper and code is detailed below which links our research decisions to the code. Examples are provided for each part of the pipeline (extraction, pre-processing, training and prediction).

The example training file (HGSM3A\_0+1\_20160304\_060000.wav) should be placed in the folder “Raw\_Data\Train”. THe example testing file (HGSM3B\_0+1\_20160308\_055700.wav) should be placed in the folder “Raw\_Data\Test”.

1. **Extract segments from the audio file**

***Python notebook****:*

Extract\_Audio.ipynb

***Paper:***

This section ties into the following part of the research paper in section:

* 3.3: “To construct the fixed-length inputs required by convolutional neural networks (CNNs), we divided each eight-hour recording into segments with window length 10s and hop length 1s (starting times of consecutive 10s segments differ by 1s). This window length was chosen so that even the longest phrase (8s) fits within a single segment; using a slightly longer segment length allows for potentially longer unseen phrases, and results in more positive segments after windowing. All audio was converted into mono.”
* 3.1: “Preprocessed amplitudes in each 10s segment were downsampled from 9600Hz to 4800Hz, and the downsampled inputs - each segment a time series of 48000 observations”.

We studied the duration of the syllables in the calls (table 1 in the supplementary material 1) and determined that using a larger segment (10 seconds) would be more appropriate as this would encapsulate calls of longer duration (more syllables or duets). Creating segments which contain complete calls would enable the neural network to tie into the ecological research question by predicting the presence or absence of a call with the segments.

***Code description:***

Extract the gibbon and non-gibbon calls from a particular .wav file. The user specifies the name of the file and the number of seconds to extract to create each of the gibbon and non-gibbon call segments. The labels need to be available (in folder 'Call\_Labels'). In this example we manually annotated the audio file. The text file which contains the annotations for the hainan gibbon call timestamps is “g\_HGSM3D\_0+1\_20160429\_051600.data” and for non-calls the file is “n\_HGSM3D\_0+1\_20160429\_051600.data”, where “g” and “n” denote gibbon and non-gibbon respectively.

The code produces two pickle output files containing the extracted segments. The default values are set to process the example file “HGSM3A\_0+1\_20160304\_060000.wav”. Ten seconds are extracted to create gibbon and non gibbon call segments.

The sampling rate is set to 4800 which downsamples from the original 9600 sampling rate. The pickled output files are saved in the “Pickled\_Data/” folder. The rationale is that reading in large 8 hour files is time consuming and thus creating pickled segmented files are more efficient for input.

Two pickle files will be saved in the folder 'Pickled\_Data', namely: 'g\_HGSM3A\_0+1\_20160304\_060000.pkl' and 'n\_HGSM3A\_0+1\_20160304\_060000.pkl'. These pickle files contain Numpy arrays which represent the segmented data. Pickle files are non-human readable and no further action is required.

From the example file 'HGSM3A\_0+1\_20160304\_060000.wav' we obtain 59 gibbon call segments and 783 non-gibbon call segments each containing 48000 data points (sample rate \* segment length, 4800 \* 10). This file originally had 15 gibbon phrases.

1. **Augmentation and converting to spectrograms (pre-processing)**

***Python notebook****:*

Augmentation.ipynb

***Paper:***

This section ties into the following part of the research paper in section

* 3.1: “We converted each audio segment into a mel-scale spectrogram, to be used as an input image to a 2-D CNN, using a window size of 1,024/9,600s, a hop size of 256/9,600s, and 128 mel frequency bins.”
* 3.1: “A bandpass filter used to extract frequencies between 1 and 2kHz”
* 3.2 “We used data augmentation to create up ten new copies of each 10s segment in both presence and absence class.”

***Code description:***

Augment the number of gibbon calls and convert both the gibbon and non-gibbon calls into spectrograms. Hyper-parameters control the amount of new data to generate.

For this example, we are processing the file 'HGSM3A\_0+1\_20160304\_060000.wav'. This file is not read from disk, instead the pickled files which were created using the 'Extract\_Audio' notebook are used. This allows for more efficiency.

The augmented data is saved to disk and the spectrograms are also saved - both as pickle files. Originally, for the example file, after applying the “Extract\_Audio” script there are 59 gibbon calls segments. After augmenting with a probability of 1 and creating 10 new files (*augmentation\_amount\_gibbon = 10*) for each original file, we obtain 590 augmented gibbon call (59 \* 10) segments. The sample rate is set to 4800 (downsampled from 9600 in the Extract\_Audio notebook). Increasing the value of the variable ‘*augmentation\_amount\_gibbon*’ will increase the number of augmented data created.

Before augmented the audio segments had 48000 points (4800 sample rate \* 10 seconds). After augmenting, the audio segments are converted to spectrograms and thus an image has the following size: 128 x 188). The code which performs this conversion is in the file 'Augmentation.py'. The number of mels is set to 128 and the hop size to 256 which results in a shape of 128 x 188.

Keras requires the depth of an image to be provided, and since the spectrograms are used as image inputs to the CNN we thus reshape the images to have a depth of 1. The final shape of the spectrograms are thus 128 x 188 x 1. The minimum and maximum frequency used when generating the spectrograms were 1Khz and 2Khz respectively - set as variables in file 'Augmentation.py'.

The spectrograms are saved to the 'Augmented\_Image\_Data' folder as two pickle files (one for the calls and the other for the non-calls). In our example these are 'g\_HGSM3A\_0+1\_20160304\_060000\_augmented\_img.pkl' and 'n\_HGSM3A\_0+1\_20160304\_060000\_augmented\_img.pkl' where 'g\_' and 'n\_' represent gibbon and non-gibbon calls respectively. Both of these files are non-human readable and require no further action.

1. **Training**

***Python notebook****:*

Train.ipynb

***Paper:***

This section ties into the following part of the research paper in section

* 3.3 “2-D CNNs use up to three convolutional layers, each followed by a max pooling layer that reduces the size of the intermediate input passed to the next layer of the network. We used 16x16 convolutional kernels 2-D CNNs. The stack of convolutional layers was followed by one or two dense layers”
* 3.3 “Each model was trained for 50 epochs using the Adam optimizer a batch size of 8 segments, and a learning rate of 0.001.”

***Code description:***

Train a CNN on the augmented spectrogram images. The model is defined in file 'CNN\_Network.py'.

The network takes image input of shape (128,188,1). Since there are usually a number of audio files used for training, these can be listed in the text file 'Training\_Files.txt’ for simplicity. Here a list of audio file names is provided (without extension). Each audio file listed in the training text file must first be processed using 'Extract\_Audio' and 'Augmentation' notebooks so that the pickled spectrogram data is available. In the case of the example provided only a single file is listed in ‘Training\_Files.txt’.

The corresponding calls and non-call data are read from the 'Augmented\_Image\_Data' folder one at a time and a message is displayed to the screen once read. The script creates the training features and targets for the CNN and begins training for a number of iterations - this is in variable '*number\_iterations = 10*'. On each iteration, the training data is used to build a new model. Thus 'number\_iterations' can be set to 1 to create 1 model, or X to create X models. Each model will perform differently given that the weights in the CNN are randomly initially on each execution. The data is split into training (80%) and validation (20%). These values are found in ‘Train\_Helper.py’.

The CNN architecture can be modified in ‘CNN\_Network.py’. The default values correspond to the 2D CNN architecture discussed in the paper.

For each model which is trained, a unique ID is created (eg: 922489) and the weights of the best model are saved in the 'Experiments' folder. The training and validation accuracy is displayed along with a confusion matrix on the validation data. The model's history is also saved for advanced Keras users. Another file is created which stores the training and validation accuracy along with the time taken for the execution. The ID numbers can be used to match all the output files.

In this example there are 1180 spectrograms (590 gibbon and 590 non-gibbon calls). The shape of X is (1180, 128, 188, 1) and Y is (1180, 2). The value of 2 is due to the fact that there are 2 classes and the labels have been encoded using 2 values [0,1] or [1,0] for gibbon and non-gibbon calls. This allows the model to produce a probabilistic distribution over the two classes.

The values of the batch size, number of epochs and class balance can be specified in the 'Hyper\_Parameters.py' file. Primarily, the batch size and class balance should be unchanged. Changing the number of epochs will impact the training performance. The default value for the number of epochs is set to 50 as per the paper.

1. **Predicting**

***Python notebook****:*

Prediction.ipynb

***Paper:***

This section ties into the following part of the research paper in section

* 3.4 “For an audio recording of arbitrary duration, our approach is to break that recording into overlapping 10s segments, and to use a trained CNN to output, for each segment starting at second s = 0, 1, 2,..., a predicted probability indicating the likelihood that at least one complete gibbon phrase is contained in the next ten seconds.”
* Table 2: “File 3 Predict: [3667, 3803], [14750, 14963], [19548, 20265], [20524, 20863]”. The script will produce this output on the test file provided.

***Code description:***

Predict on a single .wav audio file.

The model weights (from training) are saved in the 'Predictions' folder, and the name of the weights file to use is specified in the 'weights\_name' variable. In this example the weights file is '*pretrained\_weights\_from\_paper.hdf5*'. The testing file in this example is 'HGSM3B\_0+1\_20160308\_055700.wav' and the location of the test files should be in the 'Raw\_Data/Test' folder.

The same time segments which were used in training should be used when testing the model. For example, if 10 second segments were used in the 'Extract\_Audio' notebook, then the model expects 10 second inputs for prediction. This is saved in the variable 'time\_to\_extract'. The correct sampling rate should be used (same value as was used in training). All default values reflect the model’s results provided in the paper.

An output file is produced in the folder 'Predictions' which ends with 'probabilities.txt'. The format of the output is as follows: Start(seconds), End(seconds), Pr(absence), Pr(presence) - *Pr* stands for probability. By default, the model will produce an output for a window size of 10 seconds and will shift by 1 second and produce another output, and so on. The predicted components are displayed in the notebook.

In this example, the output is “[[3623, 3802], [14752, 14962], [19365, 20262], [20526, 20860]]”. The components are grouped as follows: [start time, end time]. In this example there are 4 predicted components. In this example the output file is named 'HGSM3B\_0+1\_20160308\_055700.wav\_probabilities.txt' and can be read in using spreadsheet software or otherwise. The output is comma delimited.