GDSC Soution Documentation

Two-sided approach

1. Walk through the tutorials and play with parameters

2. Create a code base that can also be used to
    - classify other animal sounds, e.g. bird songs and
    - can be used e.g. in Kaggle competitions (e.g. yearly Bird-Clef, can work w/o GPU)
    - can run on a local machine
    - implement ideas and best practices from these

Approach 1: Based on Tutorial 5

Parameters to play with:

- Learning Rate
- Sampling Rate
- Number of Epochs
- Patience Number
- Loss Function

Submission history

Submission Date	Description	Score
2023-06-17, 07:42:23	back to cross entropy loss, SR 44.1kHz, LR 2e-6, 40 epochs, patience 8	80.0937
2023-06-17, 00:00:26	epochs 40, LR 2e-6, SR 44.1kHz, metric for best model: f1	80.6022
2023-06-15, 23:50:59	HF baseline LR 2e-6 SR 44.1	81.9200

Approach 1: Findings

- Learning rate less than 1e-6 in combination with more epochs tend to overfit
- Jobs with more than 50 epochs crashed due to AWS restrictions when saving checkpoints on every epoch
- Sampling rate 44.1 kHz better than default 16 kHz (all source audios have SR of 44.1 kHz)
- High Patience numbers tend to overfit
- Cross Entropy Loss yields better results than F1-Score as metric

Approach 2: Reasons

- Reusability:
  - Able to run on local machine and in cloud environment
  - Able to be used in other environments, such as Kaggle competitions
  - Able to classify a variety of animal sounds (e.g. sounds of birds, bats, frogs)
  - Able to extend to multiclass-multilable classification
- Implement Best Practices
  - Learn from the best of the best (Kaggle competitions on bird sound classification)
  - Experiment with different pretrained models
  - Experiment with different optimizers
  - Hyperparameter tuning
  - Use a learning rate scheduler
- Experiment with audio-specific settings

Approach 2: Getting Started - AWS

- Option 1: Development in Cloud-native environments (as used in tutorials)
  - Problem: no good IDE, development/debugging very inefficient
  - AWS integration in PyCharm complicated, not achievable in reasonable time
- Option 2: Local development
  - Goal: Minimal adaption to make code run locally and in Cloud
  - Useful for migration into Cloud
  - Training/validation: local; Prediction: cloud

Approach 2: Getting Started – ML Framework

- Option 1: TensorFlow
  - Used in top-performing bird song classification [5]
  - Unable to get it to run in local non-GPU environment
- Option 2: PyTorch
  - Used in GDSC tutorials and popular in research community
  - Flexible use CPU vs. GPU
  - Easier to implement, lots of examples available

Approach 2: Experiments with Audio Settings

- Play with different time windows for audio chunks (0.5 to 60 seconds)
- Experiment with various data augmentations
- Different minimum and maximum frequencies
- Consolidate audio chunks
- How to mix white noise and background noise to the signal

Approach 2: Concept

- Modes
  1. pretrain: for hyperparameter tuning
  2. train and validate
  3. predict
- Steps
  1. load all audio data
  2. split audios into equally sized chunks
  3. upsample chunks to handle underrepresented labels
  4. create augmentation plan for chunks
  5. train, validate and predict for chunks
  6. consolidate predictions for chunks into predictions for files

Approach 2: Hyperparameter Tuning

- Python package “Optuna”
  - Easy to implement
  - State-of-the-art optimization to find parameters in a minimal amount of steps
  - Not part of the provided GDSC package
- To accelerate the process a subset of 10 classes of the overall 66 classes were used, and a maximum of 8 epochs

Approach 2: Parameters for Tuning

- Pretrained model ['efficientnet_b0', 'resnet18', 'alexnet', 'vgg16']
- Optimizer ['Adam', 'RMSprop', 'SGD']
- Learning rate and minimum learning rate (for LR scheduler)
- Target sampling rate
- Number of n-mels (for mel spectrogram; skipped for restrictions of pretrained model)
- Time windows for audio chunks
- Augmentation parameters:
  - application probability for each type of augmentations
  - maximal strength of noise vs. signal (background noise, white noise)
  - impulse response
  - jitter noise
  - time stretch and pitch shift
  - time masking and frequency masking

Approach 2: Hyperparameter Tuning Findings

- Tuning all parameters at once is by far too inefficient, hence steps taken:
  - Tuning single parameters and small groups of parameters that appear to influence each other
  - Made a decision which parameters to tune and which to be replaced by reasonable assumptions

"If there is only time to optimize one hyperparameter and one uses stochastic gradient descent, then [the learning rate] is the hyperparameter that is worth tuning." – Machine Learning Mastery [1]

Approach 2: Tuned & Skipped Hyperparameters

- Found optima:
  - Learning rate: 1.5e-3
  - Minimum learning rate (for LR scheduler): 1/20 of learning rate
  - Time windows: long window: 40 secs, focus window 2 secs (explained later)
- Skipped
  - Pretrained model: All bird song recognition algorithms use EfficientNet_B0 [6]
  - Optimizer: Adam is the fastest converging algorithm
  - Audio Sample Rate: used 44.1 kHz (so that there is no loss of crucial information)
  - n_mels for Mel-Spectrogram: EfficientNet_B0 requires input of 3x224x224
  - augmentation related parameters
  - maximum and minumum frequencies (maximum set to half the sample rate, however played with various minimum frequencies between 1/32 and 1/64 of the SR)

Approach 2: Learning Rate Scheduler [2]

State-of-the-art learning rate scheduler that is used for bird song recognition:

CosineAnnealingLR

Required parameters:

- Initial learning rate
- Minimum learning rate

Approach 2: Time Windows

Special patterns appear in the spectrogram for grasshoppers and cicada on long time windows (chirp vs. no chirp) and on very short time windows (aka focus window), showing the way they chirp. Examples from 60s time window:

Examples from 0.6s time window:

In order to make use of all the information, spectrograms of long time window and focus window were combined:

Approach 2: Combined Spectrograms

Examples of combined spectrograms. In some cases only the long time or the focus window provide distinct patterns

long: 60s, focus 0.6s

long: 40s, focus 1s

long 16s (top) with four foci à 1s

long 40s (top) with four foci à 2s

Approach 2: Selection of Focus Window

Depending on how many foci I want to use I would split the long time window into that number of chunks, then take the absolute values of the audio data, calculate the rolling average of the focus window, and select the one with the highest value (to make sure to not just have noise there)

Detailed view of last example from previous slide (40s top + 4x2s)

Approach 2: Augmentations

Applied augmentations:

1. Background noise: Taken from publicly available sources, some taken from youtube (e.g. wind and rain). Mixed to signal at maximal strength between 20 and 40%, probability 80%
2. White noise: Mixed to signal at maximal strength of 20%, probability 50%
3. Jitter: Sampling jitter occurs when audios are resampled. Since a part of the audio files have been downsampled to 44.1 kHz, about 25% have been resampled a few times and finally back to 44.1 kHz
4. Time stretch: Maximal stretch +/-2.5%, probability 70%
5. Pitch shift: Maximally 0.5 semitones, probability 70%

Skipped:

1. Extracting signals from spectrograms by applying median filtering + overall median / time median / frequency median (caused inf/NaN values in tensors)
2. Time masking and frequency masking (because (1) did not work properly)

Approach 2: Miscellaneous

Upsampling: In order to deal with class imbalances, classes with a low number of occurrences have been upsampled to a certain amount (number of configurable, experimented with values between 40 and 80)

Augmentation plan: After splitting the audio data into chunks and upsampling, for every chunk it will be determined which augmentation to apply

Train-Val-split: combine train and val dataset and perform stratified split

Consolidation of predictions of chunks to predictions of files, based on the probabilities of the classes (labels): Easy ways such as building the average of chunks or selecting the maximum value yielded inferior results, hence transformations have been applied in a way that two 80% probabilities are about equal to one 90% probability

Local vs. Cloud: In order to make the locally developed code compatible with cloud (AWS), the code requires a few changes: parsing some arguments (such as input source and output destination), and set code related to the optuna package to comment (pretraining only occurs locally, predictions only occur in the cloud and do not require that package)

Approach 2: First Results

Various time windows yielded mixed results, between 71 and 83.

The submission titled “final submission” uses the best result (4 times focus 2 secs) and the result from approach 1 by applying the ensemble method of voting.

At this time I was supposed to be finished (rank 24 at that time)

Submission Date	Description	Score
2023-07-09, 06:13:13	final submission	84.9574
2023-07-09, 05:41:17	ensembles on details	82.1126
2023-07-09, 03:37:12	4 times focus 2 secs	82.8362
2023-07-07, 23:05:23	time frame 60s focus .6s	71.3299
2023-07-04, 07:01:53	time frame 40s focus 1s	79.2133

Approach 2: Next Attempt: Colors

The idea of using colors for different frequency bands came up during office hours, I wanted to give it a shot. Also, I wanted to arrange several focus windows below a thinner long time window.

Examples (Four octaves projected to RGB channels, 40s + 4x2s):

Approach 2: Second Results

Results using colors were disappointing, however using more focus windows seemed to yield better results. My reasoning was that pushing that to the extreme might achieve even better results.

Submission Date	Description	Score
2023-07-12, 22:46:25	window 16 focus 1	83.9754
2023-07-11, 00:01:49	rgb 4o	70.0243

One last attempt before the deadline with only 1 second focus windows, Result also disappointing.

Submission Date	Description	Score
2023-07-15, 16:08:47	chunks 1 sec	73.7027

Approach 2: Finally...

To close things up I decided to do a weighted voting on all previous results, including the poor results, the results being the weights.

Submission Date	Description	Score
2023-07-15, 23:50:34	voted	91.6294

Huge surprise:
Creating a strong prediction from several inferior predictions can be powerful.

Things to Try

Given more time, there would have been more things I'd like to try:

- Examine each type of data augmentation and their impact on the score, and choose accordingly
- Make attempts "normalizing" the spectrograms using the 2d-median filtering + median per time and frequency or overall median
- Make attempts to extract signal from noise (would be a good attempt, if the signal is clear to identify)
- Create an additional class for "noise only" that will not be considered when consolodating the predictions of audio chunks into predictions for the files
- Use machine learning approach to consolidate classification for audio chunks into classification for original files
- Dedicate more time to tune hyperparameters

Thank You

Leaderboard Validation

Rank	Team Name	Validation Score
1	It is a bug	92.0271
2	Gandalf the Data-Wise and Aragorn the Code-Master	90.9662
3	HiveMind Data Insectors	89.7086
4	BuzzAstral	89.5362
5	The Buzz Group	89.3105
6	gracehopper	89.2014
7	ANTificial neural network	88.7356

Appendix

Code to make program cloud (AWS) compatible:

```
parser = argparse.ArgumentParser()
parser.add_argument("--data_channel", type=str, default=os.environ["SM_CHANNEL_DATA"])
parser.add_argument("--test_dir", type=str, default="test")
parser.add_argument("--output_dir", type=str, default=os.environ['SM_MODEL_DIR'])
args, _ = parser.parse_known_args()
output_dir = f"{args.output_dir}/"
input_root = f"{args.data_channel}/"
```

Relevant programs and jupyter notebooks:

- baseline_ast_train: based on tutorial; result 81.96%
- train_2: long time window: 40 sec, focus window: 1 sec; result 79.21%
- train_2: long time window: 60 sec, focus window: 0.6 sec; result 71.33%
- train_2: long time window: 16 sec, four focus windows: 2 sec; result 82.84%
- train_3: long time window: 40 sec, four focus windows: 2 sec, using RGB for 4 octaves; result 70.23%
- train_4: long time window: 16 sec, four focus windows: 1 sec; result 83.98% (accidently not used in voting)
- train_6: long/short time window: 1.25 sec; result 73.70%
- train_x: cloud version, copied and adapted from train_2/3/4/5/6
- voting: combine previous results; result

other:

- train_5: long time window: 2.4 sec, focus windows: 2 sec (cancelled due to time constraints)

Extreme example for sampling jitter:

Sources

[1] Hyperparameter Tuning: https://machinelearningmastery.com/recommendations-for-deep-learning-neural-network-practitioners/

[2] Learning rate schedulers: https://towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863

[3] Image classification tutorial for beginners: https://towardsdatascience.com/pytorch-image-classification-tutorial-for-beginners-94ea13f56f2

[4] Multiclass Image Classification with PyTorch: https://medium.com/analytics-vidhya/multiclass-image-classification-with-pytorch-af7578e10ee6

[5] “BirdCLEF23: Pretraining is all you need”: https://www.kaggle.com/code/awsaf49/birdclef23-pretraining-is-all-you-need-train

[6] Comparison of Pretrained Models: https://www.mdpi.com/2224-2708/10/4/72