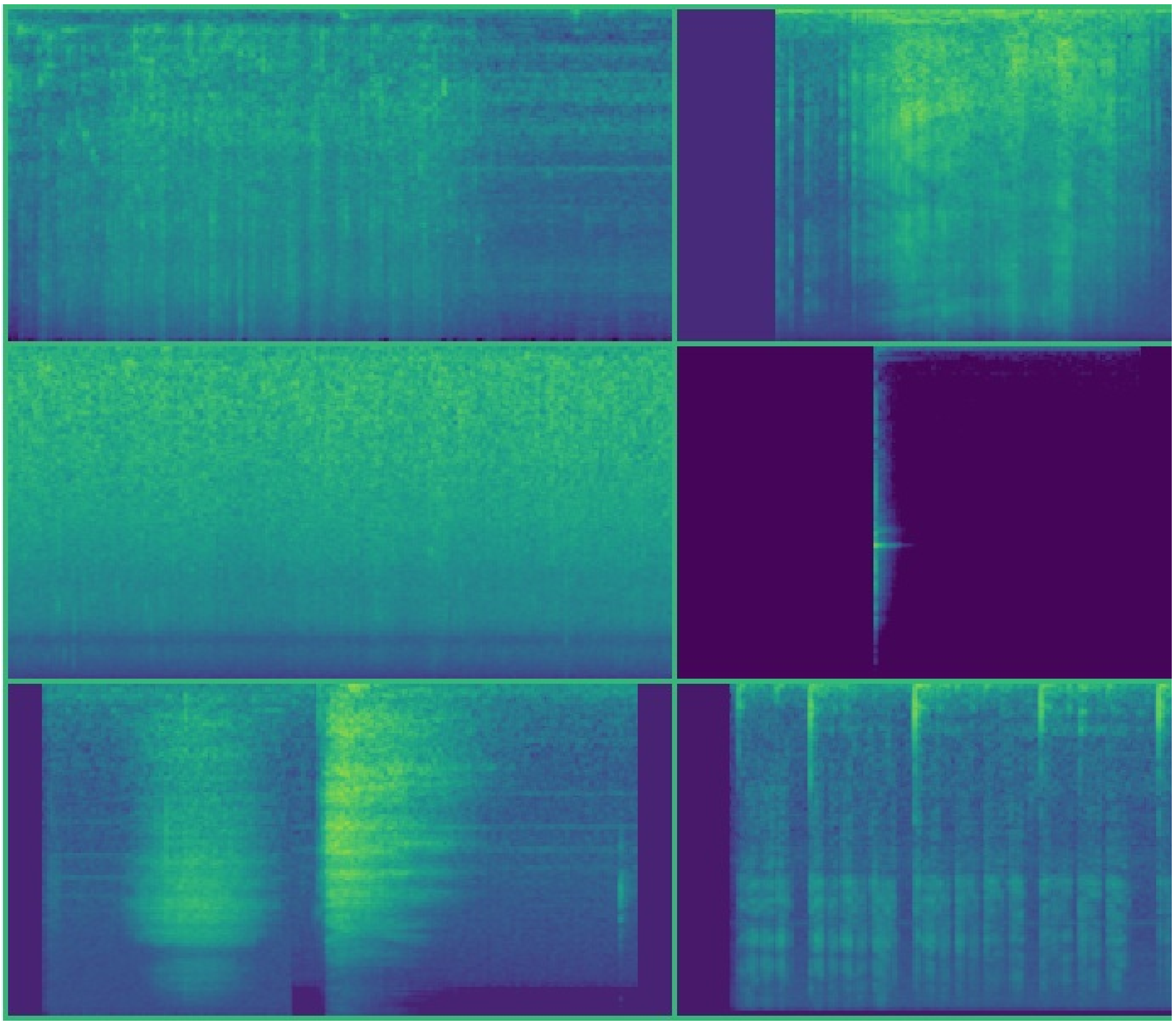
Argus solution Freesound Audio Tagging 2019



This repo contains the source code of the 1st place solution for Freesound Audio Tagging 2019 Challenge. The goal of the competition is to develop an algorithm for automated multi-label audio tagging. The main research problem of this competition is to properly utilize a small amount of reliable, manually-labeled data, and a larger quantity of noisy audio data from the web in a multi-label classification task with a large vocabulary (80 categories).

Solution

Key points: *Log-scaled mel-spectrograms* CNN model with attention, skip connections and auxiliary classifiers *SpecAugment, Mixup augmentations* Hand relabeling of the curated dataset samples with a low score * Ensembling with an MLP second-level model and a geometric mean blending

The Argus framework for PyTorch was employed. It makes the learning process more straightforward and the code briefer.

Data preprocessing

Log-scaled mel-spectrograms is the modern standard way of the data representation in CNN-based audio scene classification. Converting audio to spectrograms in this solution was inspired by the daisukelab's data preprocessing notebook. Audio config parameters:

```
sampling_rate = 44100
hop_length = 345 * 2
fmin = 20
fmax = sampling_rate // 2
n_mels = 128
n_fft = n_mels * 20
min_seconds = 0.5
```

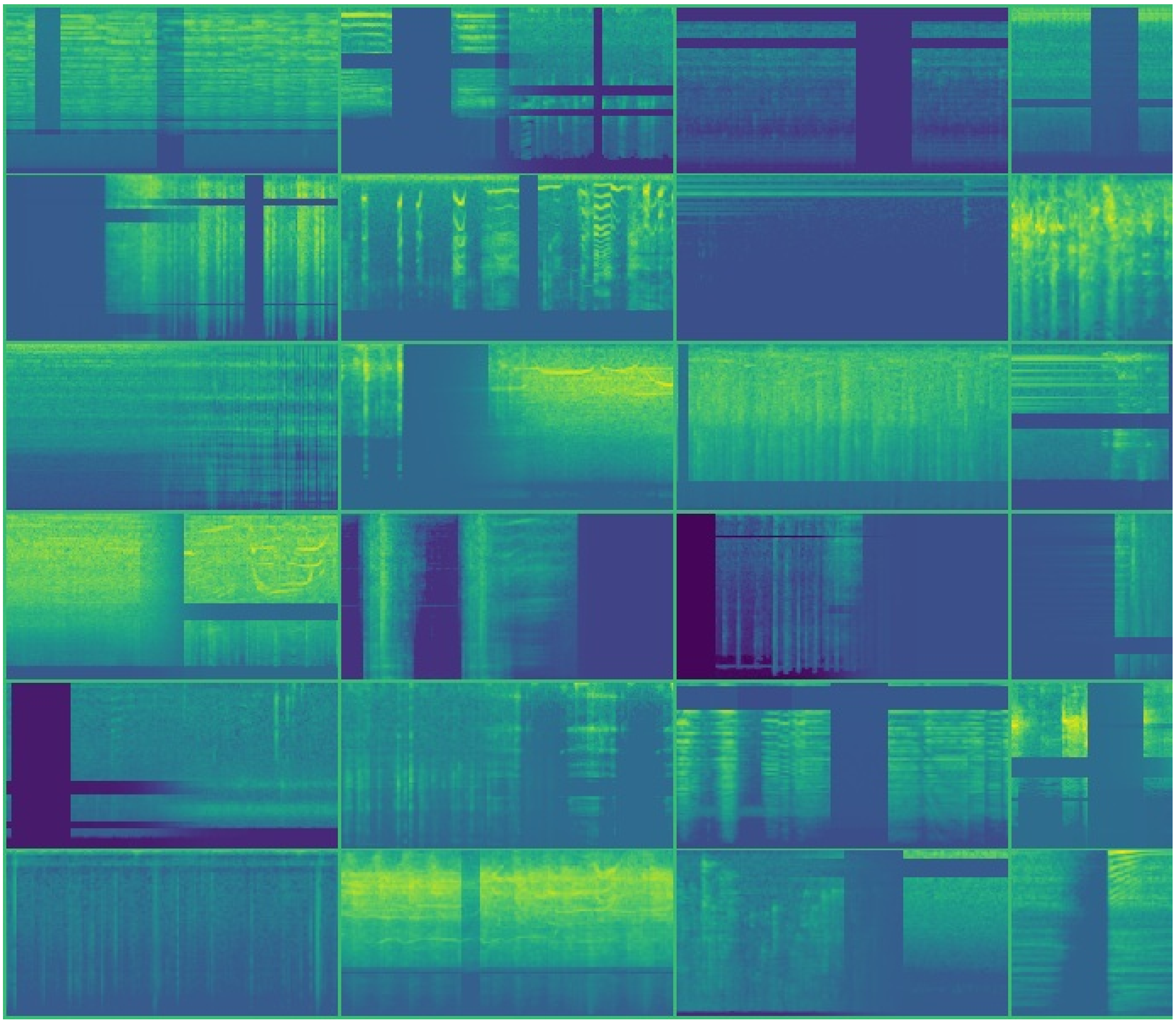
Augmentations

Several augmentations were applied on spectrograms during the training stage. The part of transforms.py lists augmentation techniques:

```
size = 256
transforms = Compose([
    OneOf([
        PadToSize(size, mode='wrap'),      # Reapeat small clips
        PadToSize(size, mode='constant'),  # Pad with a minimum value
    ], p=[0.5, 0.5]),
    RandomCrop(size),                      # Crop 256 values on time axi
    UseWithProb(
        # Random resize crop helps a lot, but I can't explain why ¯\_(ツ
        RandomResizedCrop(scale=(0.8, 1.0), ratio=(1.7, 2.3)),
        prob=0.33
    ),
    # SpecAugment [1], masking blocks of frequency channels, and masking
    UseWithProb(SpecAugment(num_mask=2,
                            freq_masking=0.15,
                            time_masking=0.20), 0.5),
    # Use librosa.feature.delta with order 1 and 2 for creating 2 additi
    # then divide by 100
    ImageToTensor()
])
```

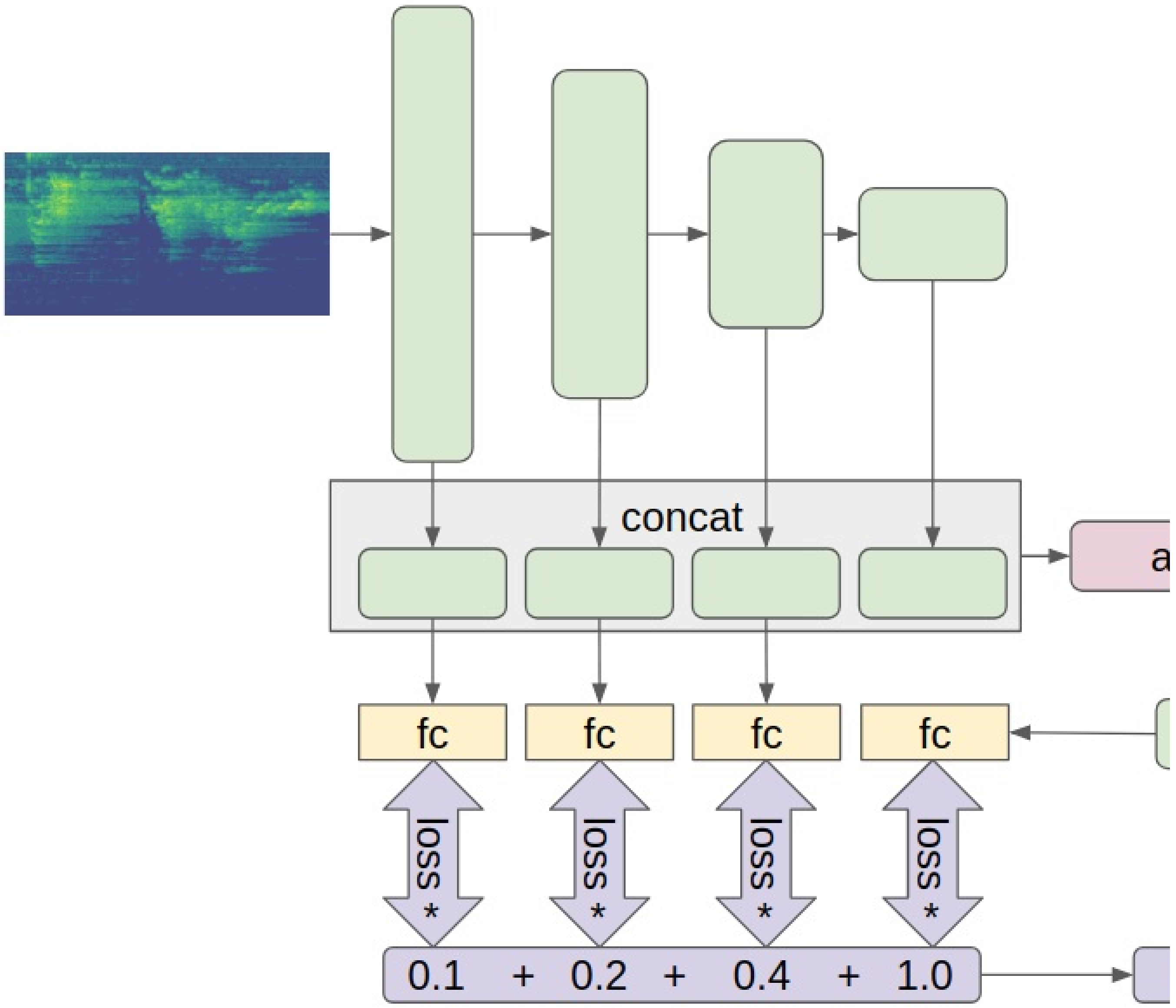
MixUp [2] augmentation was found to be beneficial in the competition. This method creates a new training sample based on the weighted average of two items from the original dataset. Additionally, SigmoidConcatMixer was applied. It produces a merged sample with a smooth (sigmoid-based) transition from one audio-clip to another over time.

There are some augmented spectrograms, and they look crazy :)



Model

Model from [mhiro2's kernel]() was used as a starting point. After numerous experiments, the original architecture was modified with attention, skip connections, and auxiliary classifiers.



Training

- 5 random folds
- Loss: BCE on curated, Lsoft [3] with beta 0.7 on noisy data
- Optimizer: Adam with initial LR 0.0009
- LR scheduler: Reduce on a plateau with patience 6, factor 0.6
- Use different probabilities for sampling curated and noisy data
- Training on hand relabeled curated samples with a low lwlrap score by previous models
- Training with BCE on noisy samples with a high lwlrap score by previous models
- Mixed precision training with apex.amp allows using batch size 128 with input size 256x128 px

Ensemble

The geometric mean of `7` first-level models and `3` second-level models was used for the final submission. MLPs trained with different hyperparameters were used as second-level models. Seven first-level models were chosen by enumeration of combinations of training experiments to finding the highest CV score.

Lab journal

The progress of the solution during the competition can be seen in the laboratory journal. It describes all the experiments and ideas, but it is partially in Russian, sorry :).

Quick setup and start

Requirements

Software

- Linux
- Nvidia drivers, CUDA `>= 10.0`, cuDNN `>= 7`
- Docker, nvidia-docker

The provided Dockerfile is supplied to build an image with CUDA support and cuDNN.

Hardware

- `32`GB of RAM
- `2080`ti or another GPU with fp`16` support and at least `12`GB memory

Preparations

- Clone the repo, build docker image. `bash git clone https://github.com/lRomul/argus-freesound.git cd argus-freesound make build`

- Download and extract dataset to `data` folder

  Folder structure should be: `data ├── README.md ├── sample_submission.csv ├── test ├── train_curated ├── train_curated.csv ├── train_noisy └── train_noisy.csv`

Run

- Run docker container `bash make run`

- Create a file with folds split `bash python make_folds.py`

Single model

For example, take the experiment `corr_noisy_008`, which currently is in the train_folds.py:

- Train single `5` fold model

`bash python train_folds.py --experiment corr_noisy_008`

Model weights will be in `data/experiments/corr_noisy_008`

- Predict train and test, evaluate metrics

  `bash python predict_folds.py --experiment corr_noisy_008`

Predictions, submission file, and validation metrics will be saved in `data/predictions/corr_noisy_008`

Ensemble

If you want to reproduce the whole ensemble, you should train all experiments in stacking_predict.py, script ensemble_pipeline.sh can help:

- Download and extract data.
- Run full pipeline `bash ./ensemble_pipeline.sh`
- Kernel code will be saved in `kernel/stacking_kernel_template.py`.
- Models weights will be saved in `data/experiments`. You can zip `experiments` folder and upload the archive to kaggle dataset.

Kernel build system

It was quite challenging to manage the project without a way to split the solution into modules. The idea of kernel building from the first place solution of the Mercari Price Suggestion Challenge was used. You can find the build system template here. To create a submission, run `python build_kernel.py`, this would compress the whole project into scripts in the `kernel` folder:
`kernel_template.py` - single model submission
`stacking_kernel_template.py` - ensemble submission

References

[1] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, Quoc V. Le, "*SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition*", arXiv:1904.08779, 2019.

[2] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz, "*mixup: Beyondempirical risk minimization*", arXiv:1710.09412, 2017.

[3] Eduardo Fonseca, Manoj Plakal, Daniel P. W. Ellis, Frederic Font, Xavier Favory, Xavier Serra, "*Learning Sound Event Classifiers from Web Audio with Noisy Labels*", arXiv:1901.01189, 2019.