Update

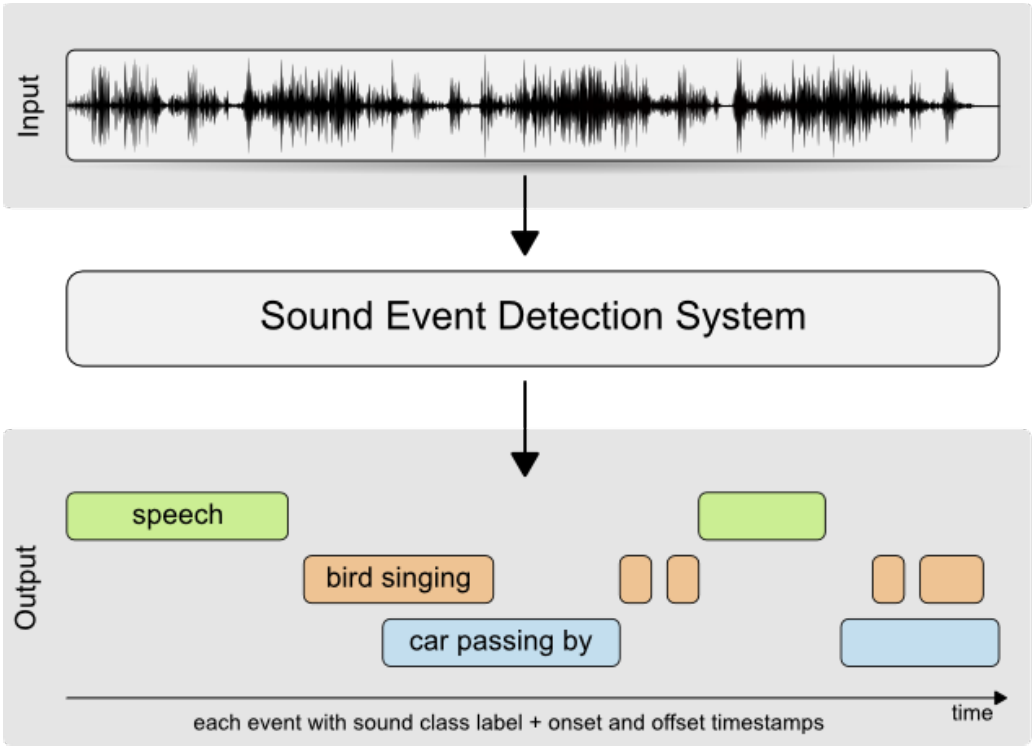
Version note

v3:

- Training procedure didn't use pretrained weight. Changed to use pretrained weight.
- Add link to original paper.

About

In this notebook, I will introduce Sound Event Detection (SED) task and model fit for that task, and I will show how to train SED model with only *weak* annotation.



In SED task, we need to detect sound events from continuous (long) audio clip, and provide prediction of *what sound event exists from when to when*. Therefore our prediction for SED task should look like this.

clip_id	onset	offset	event
audio_001	0.952	1.87	car
audio_001	3.82	6.75	speech
audio_001	5.32	9.28	alarm

SED task is different from the tasks in past audio competitions in kaggle. The task in Freesound Audio Tagging 2019 (https://www.kaggle.com/c/freesound-audio-tagging-2019) or Freesound General-Purpose Audio Tagging Challenge (https://www.kaggle.com/c/freesound-audio-tagging) is Audio Tagging, which we'll need to provide clip level prediction, and the task in TensorFlow Speech Recognition Challenge (https://www.kaggle.com/c/tensorflow-speech-recognition-challenge) is Speech Recognition, so what we need to predict is which speech command is in that audio clip (which is in a sense similar to Audio Tagging task, because we only need to provide clip level prediction).

In this competition, what we need to provide is 5sec chunk level prediction for `site_1` and `site_2` data, and clip level prediction for `site_3` data. Chunk level prediction can be treated as audio tagging task if we treat each chunk as short audio clip, but we can also use SED approach.

Model for SED task

How can we provide prediction with `onset` and `offset` time information? To do this, models for SED task output *segment-wise* prediction instead of outputting aggregated prediction for a clip, which is usually used for Audio Tagging model.

How can we output *segment-wise* prediction? The idea is simple. Assume we use 2D CNN based model which takes log-melspectrogram as input and extract features using CNN feature extractor, and do classification with the feature map which is the output of CNN.

The output of CNN feature extractor still contains information about frequency and time(it should be 4 dimensional: (batch size, channels, frequency, time)), so if we aggregate it only in frequency axis, we can preserve time information on that feature map. That feature map has information about which time segment has what sound event.

Now that I've introduced the basic idea, let's look into a SED model with some code. In this notebook, I'll use (Weakly-supervised) SED model provided by PANNs repository (https://github.com/qiuqiangkong/audioset_tagging_cnn/). The model here is pretrained with AudioSet (https://research.google.com/audioset/), which is an ImageNet counterpart in audio field.

PANNs paper (https://arxiv.org/abs/1912.10211)

In [1]:
```
import cv2
import audioread
import logging
import os
import random
import time
import warnings

import librosa
import librosa.display as display
import numpy as np
import pandas as pd
import soundfile as sf
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data

from contextlib import contextmanager
from IPython.display import Audio
from pathlib import Path
from typing import Optional, List

from catalyst.dl import SupervisedRunner, State, CallbackOrder, Callback, Ch
eckpointCallback
from fastprogress import progress_bar
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score, average_precision_score
```

```
wandb: WARNING W&B installed but not logged in.  Run `wandb login` or set the
WANDB_API_KEY env variable.
/opt/conda/lib/python3.7/site-packages/tqdm/std.py:666: FutureWarning: The Pa
nel class is removed from pandas. Accessing it from the top-level namespace w
ill also be removed in the next version
  from pandas import Panel
```

In [2]:
```
def set_seed(seed: int = 42):
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)  # type: ignore
    torch.backends.cudnn.deterministic = True  # type: ignore
    torch.backends.cudnn.benchmark = True  # type: ignore


def get_logger(out_file=None):
    logger = logging.getLogger()
    formatter = logging.Formatter("%(asctime)s - %(levelname)s - %(message)s
")
    logger.handlers = []
    logger.setLevel(logging.INFO)

    handler = logging.StreamHandler()
    handler.setFormatter(formatter)
    handler.setLevel(logging.INFO)
    logger.addHandler(handler)

    if out_file is not None:
        fh = logging.FileHandler(out_file)
        fh.setFormatter(formatter)
        fh.setLevel(logging.INFO)
        logger.addHandler(fh)
    logger.info("logger set up")
    return logger


@contextmanager
def timer(name: str, logger: Optional[logging.Logger] = None):
    t0 = time.time()
    msg = f"[{name}] start"
    if logger is None:
        print(msg)
    else:
        logger.info(msg)
    yield

    msg = f"[{name}] done in {time.time() - t0:.2f} s"
    if logger is None:
        print(msg)
    else:
        logger.info(msg)


set_seed(1213)
```

In [3]:
```
ROOT = Path.cwd().parent
INPUT_ROOT = ROOT / "input"
RAW_DATA = INPUT_ROOT / "birdsong-recognition"
TRAIN_AUDIO_DIR = RAW_DATA / "train_audio"
TRAIN_RESAMPLED_AUDIO_DIRS = [
  INPUT_ROOT / "birdsong-resampled-train-audio-{:0>2}".format(i)  for i in r
ange(5)
]
TEST_AUDIO_DIR = RAW_DATA / "test_audio"
```

In [4]:
```
train = pd.read_csv(TRAIN_RESAMPLED_AUDIO_DIRS[0] / "train_mod.csv")

if not TEST_AUDIO_DIR.exists():
    TEST_AUDIO_DIR = INPUT_ROOT / "birdcall-check" / "test_audio"
    test = pd.read_csv(INPUT_ROOT / "birdcall-check" / "test.csv")
else:
    test = pd.read_csv(RAW_DATA / "test.csv")
```

torchlibrosa

In PANNs, `torchlibrosa`, a PyTorch based implementation are used to replace some of the `librosa`'s functions. Here I use some functions of `torchlibrosa`.

Ref: https://github.com/qiuqiangkong/torchlibrosa (https://github.com/qiuqiangkong/torchlibrosa)

LICENSE

In [5]:
```
class DFTBase(nn.Module):
    def __init__(self):
        """Base class for DFT and IDFT matrix"""
        super(DFTBase, self).__init__()

    def dft_matrix(self, n):
        (x, y) = np.meshgrid(np.arange(n), np.arange(n))
        omega = np.exp(-2 * np.pi * 1j / n)
        W = np.power(omega, x * y)
        return W

    def idft_matrix(self, n):
        (x, y) = np.meshgrid(np.arange(n), np.arange(n))
        omega = np.exp(2 * np.pi * 1j / n)
        W = np.power(omega, x * y)
        return W


class STFT(DFTBase):
    def __init__(self, n_fft=2048, hop_length=None, win_length=None,
        window='hann', center=True, pad_mode='reflect', freeze_parameters=Tr
ue):
        """Implementation of STFT with Conv1d. The function has the same out
put
        of librosa.core.stft
        """
        super(STFT, self).__init__()

        assert pad_mode in ['constant', 'reflect']

        self.n_fft = n_fft
        self.center = center
        self.pad_mode = pad_mode

        # By default, use the entire frame
        if win_length is None:
            win_length = n_fft

        # Set the default hop, if it's not already specified
        if hop_length is None:
            hop_length = int(win_length // 4)

        fft_window = librosa.filters.get_window(window, win_length, fftbins=
True)

        # Pad the window out to n_fft size
        fft_window = librosa.util.pad_center(fft_window, n_fft)

        # DFT & IDFT matrix
        self.W = self.dft_matrix(n_fft)

        out_channels = n_fft // 2 + 1

        self.conv_real = nn.Conv1d(in_channels=1, out_channels=out_channels,
            kernel_size=n_fft, stride=hop_length, padding=0, dilation=1,
            groups=1, bias=False)

        self.conv_imag = nn.Conv1d(in_channels=1, out_channels=out_channels,
            kernel_size=n_fft, stride=hop_length, padding=0, dilation=1,
            groups=1, bias=False)

        self.conv_real.weight.data = torch.Tensor(
            np.real(self.W[:, 0 : out_channels] * fft_window[:, None]).T)[:,
None, :]
        # (n_fft // 2 + 1, 1, n_fft)
```

In [6]:
```
class DropStripes(nn.Module):
    def __init__(self, dim, drop_width, stripes_num):
        """Drop stripes.
        Args:
          dim: int, dimension along which to drop
          drop_width: int, maximum width of stripes to drop
          stripes_num: int, how many stripes to drop
        """
        super(DropStripes, self).__init__()

        assert dim in [2, 3]    # dim 2: time; dim 3: frequency

        self.dim = dim
        self.drop_width = drop_width
        self.stripes_num = stripes_num

    def forward(self, input):
        """input: (batch_size, channels, time_steps, freq_bins)"""

        assert input.ndimension() == 4

        if self.training is False:
            return input

        else:
            batch_size = input.shape[0]
            total_width = input.shape[self.dim]

            for n in range(batch_size):
                self.transform_slice(input[n], total_width)

            return input


    def transform_slice(self, e, total_width):
        """e: (channels, time_steps, freq_bins)"""

        for _ in range(self.stripes_num):
            distance = torch.randint(low=0, high=self.drop_width, size=(1,))
[0]
            bgn = torch.randint(low=0, high=total_width - distance, size=(1,
))[0]

            if self.dim == 2:
                e[:, bgn : bgn + distance, :] = 0
            elif self.dim == 3:
                e[:, :, bgn : bgn + distance] = 0


class SpecAugmentation(nn.Module):
    def __init__(self, time_drop_width, time_stripes_num, freq_drop_width,
        freq_stripes_num):
        """Spec augmetation.
        [ref] Park, D.S., Chan, W., Zhang, Y., Chiu, C.C., Zoph, B., Cubuk,
E.D.
        and Le, Q.V., 2019. Specaugment: A simple data augmentation method
        for automatic speech recognition. arXiv preprint arXiv:1904.08779.
        Args:
          time_drop_width: int
          time_stripes_num: int
          freq_drop_width: int
          freq_stripes_num: int
        """

        super(SpecAugmentation, self).__init__()
```

audioset_tagging_cnn

I also use `Cnn14_DecisionLevelAtt` model from PANNs models (https://github.com/qiuqiangkong/audioset_tagging_cnn/blob/master/pytorch/models.py), which is a SED model.

LICENSE

Building blocks

In [7]:
```
def init_layer(layer):
    nn.init.xavier_uniform_(layer.weight)

    if hasattr(layer, "bias"):
        if layer.bias is not None:
            layer.bias.data.fill_(0.)


def init_bn(bn):
    bn.bias.data.fill_(0.)
    bn.weight.data.fill_(1.0)


def interpolate(x: torch.Tensor, ratio: int):
    """Interpolate data in time domain. This is used to compensate the
    resolution reduction in downsampling of a CNN.

    Args:
      x: (batch_size, time_steps, classes_num)
      ratio: int, ratio to interpolate
    Returns:
      upsampled: (batch_size, time_steps * ratio, classes_num)
    """
    (batch_size, time_steps, classes_num) = x.shape
    upsampled = x[:, :, None, :].repeat(1, 1, ratio, 1)
    upsampled = upsampled.reshape(batch_size, time_steps * ratio, classes_nu
m)
    return upsampled


def pad_framewise_output(framewise_output: torch.Tensor, frames_num: int):
    """Pad framewise_output to the same length as input frames. The pad valu
e
    is the same as the value of the last frame.
    Args:
      framewise_output: (batch_size, frames_num, classes_num)
      frames_num: int, number of frames to pad
    Outputs:
      output: (batch_size, frames_num, classes_num)
    """
    pad = framewise_output[:, -1:, :].repeat(
        1, frames_num - framewise_output.shape[1], 1)
    """tensor for padding"""

    output = torch.cat((framewise_output, pad), dim=1)
    """(batch_size, frames_num, classes_num)"""

    return output


class ConvBlock(nn.Module):
    def __init__(self, in_channels: int, out_channels: int):
        super().__init__()

        self.conv1 = nn.Conv2d(
            in_channels=in_channels,
            out_channels=out_channels,
            kernel_size=(3, 3),
            stride=(1, 1),
            padding=(1, 1),
            bias=False)

        self.conv2 = nn.Conv2d(
            in_channels=out_channels,
            out_channels=out_channels,
            kernel_size=(3, 3),
```

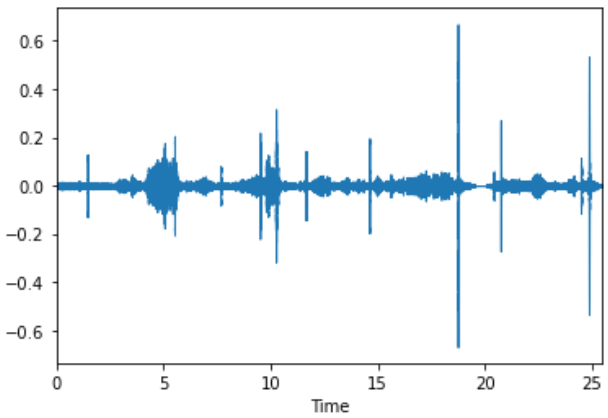
In [8]:
```
class PANNsCNN14Att(nn.Module):
    def __init__(self, sample_rate: int, window_size: int, hop_size: int,
                 mel_bins: int, fmin: int, fmax: int, classes_num: int):
        super().__init__()

        window = 'hann'
        center = True
        pad_mode = 'reflect'
        ref = 1.0
        amin = 1e-10
        top_db = None
        self.interpolate_ratio = 32  # Downsampled ratio

        # Spectrogram extractor
        self.spectrogram_extractor = Spectrogram(
            n_fft=window_size,
            hop_length=hop_size,
            win_length=window_size,
            window=window,
            center=center,
            pad_mode=pad_mode,
            freeze_parameters=True)

        # Logmel feature extractor
        self.logmel_extractor = LogmelFilterBank(
            sr=sample_rate,
            n_fft=window_size,
            n_mels=mel_bins,
            fmin=fmin,
            fmax=fmax,
            ref=ref,
            amin=amin,
            top_db=top_db,
            freeze_parameters=True)

        # Spec augmenter
        self.spec_augmenter = SpecAugmentation(
            time_drop_width=64,
            time_stripes_num=2,
            freq_drop_width=8,
            freq_stripes_num=2)

        self.bn0 = nn.BatchNorm2d(mel_bins)

        self.conv_block1 = ConvBlock(in_channels=1, out_channels=64)
        self.conv_block2 = ConvBlock(in_channels=64, out_channels=128)
        self.conv_block3 = ConvBlock(in_channels=128, out_channels=256)
        self.conv_block4 = ConvBlock(in_channels=256, out_channels=512)
        self.conv_block5 = ConvBlock(in_channels=512, out_channels=1024)
        self.conv_block6 = ConvBlock(in_channels=1024, out_channels=2048)

        self.fc1 = nn.Linear(2048, 2048, bias=True)
        self.att_block = AttBlock(2048, classes_num, activation='sigmoid')

        self.init_weight()

    def init_weight(self):
        init_bn(self.bn0)
        init_layer(self.fc1)

    def cnn_feature_extractor(self, x):
        x = self.conv_block1(x, pool_size=(2, 2), pool_type='avg')
        x = F.dropout(x, p=0.2, training=self.training)
        x = self.conv_block2(x, pool_size=(2, 2), pool_type='avg')
        x = F.dropout(x, p=0.2, training=self.training)
        x = self.conv_block3(x, pool_size=(2, 2), pool_type='avg')
```

What is good in PANNs models is that they accept raw audio clip as input. Let's put a chunk into the CNN feature extractor of the model above.

In [9]:
```
SR = 32000

y, _ = librosa.load(TRAIN_RESAMPLED_AUDIO_DIRS[0] / "aldfly" / "XC134874.wav
",
                    sr=SR,
                    res_type="kaiser_fast",
                    mono=True)

Audio(y, rate=SR)
```

Out[9]:

In [10]:
```
display.waveplot(y, sr=SR);
```



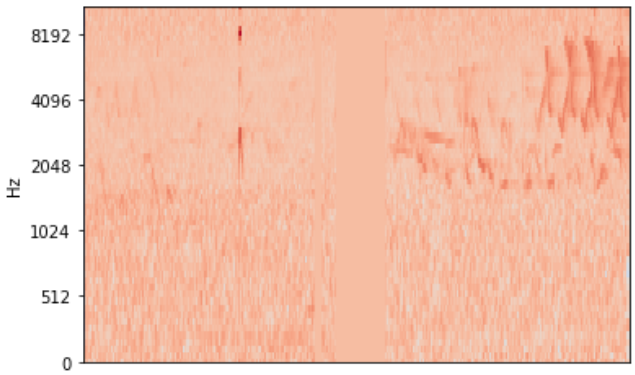
In [11]:
```
model_config = {
    "sample_rate": 32000,
    "window_size": 1024,
    "hop_size": 320,
    "mel_bins": 64,
    "fmin": 50,
    "fmax": 14000,
    "classes_num": 264
}

model = PANNsCNN14Att(**model_config)
```

In `PANNsCNN14Att`, input raw waveform will be converted into log-melspectrogram using `torchlibrosa`'s utilities. I put this functionality in `PANNsCNN14Att.preprocess()` method. Let's check the output.

In [12]:
```
chunk = torch.from_numpy(y[:SR * 5]).unsqueeze(0)
melspec, _ = model.preprocess(chunk)
melspec.size()
```

Out[12]:
```
torch.Size([1, 1, 501, 64])
```

In [13]:
```
melspec_numpy = melspec.detach().numpy()[0, 0].transpose(1, 0)
display.specshow(melspec_numpy, sr=SR, y_axis="mel");
```



`PANNsCNN14Att.cnn_feature_extractor()` method will take this as input and output feature map. Let's check the output of the feature extractor.

In [14]:
```
feature_map = model.cnn_feature_extractor(melspec)
feature_map.size()
```

Out[14]:
```
torch.Size([1, 2048, 15, 2])
```

Although it's downsized through several convolution and pooling layers, the size of it's third dimension is 15 and it still contains time information. Each element of this dimension is *segment*. In SED model, we provide prediction for each of this.

Train SED model with only weak supervision

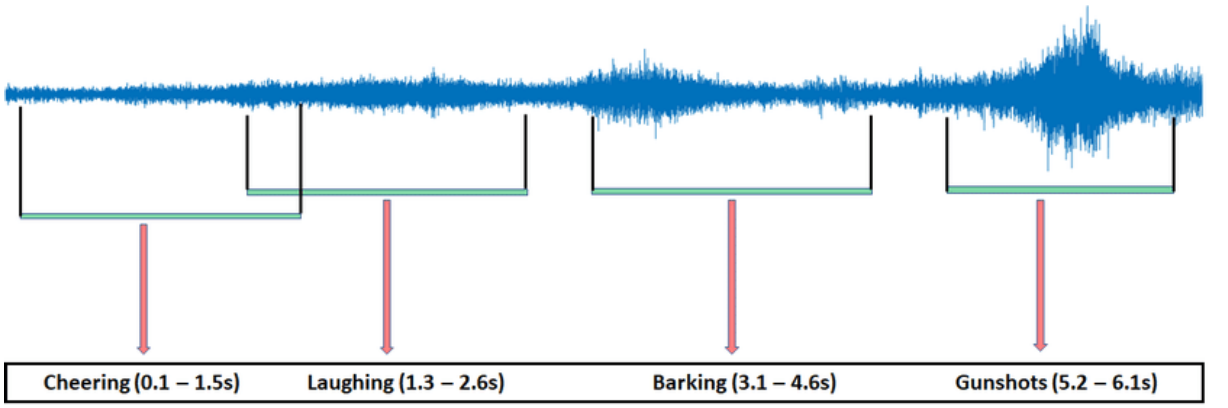


Illustration of Strong Labeling of Events

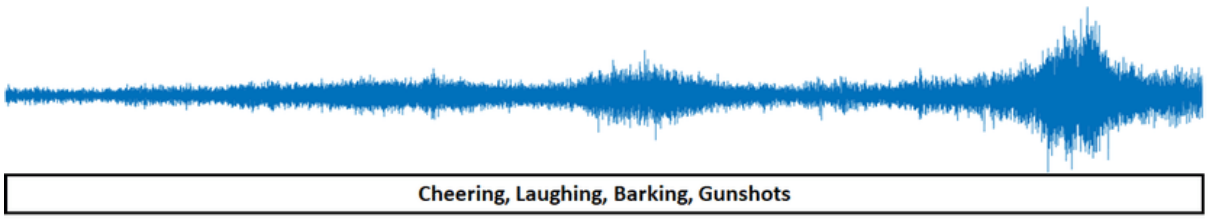


Illustration of Weak Labeling of Events

This figure gives us an intuitive explanation what is *weak annotation* and what is *strong annotation* in terms of sound event detection. For this competition, we only have weak annotation (clip level annotation). Therefore, we need to train our SED model in weakly-supervised manner.

In weakly-supervised setting, we only have clip-level annotation, therefore we also need to aggregate that in time axis. Hense, we at first put classifier that outputs class existence probability for each time step just after the feature extractor and then aggregate the output of the classifier result in time axis. In this way we can get both clip-level prediction and segment-level prediction (if the time resolution is high, it can be treated as event-level prediction). Then we train it normally by using BCE loss with clip-level prediction and clip-level annotation.

Let's check how this is implemented in the PANNs model above. segment-wise prediction and clip-wise prediction is actually calculated in `AttBlock` of the model.

```
class AttBlock(nn.Module):
    def __init__(self,
                 in_features: int,
                 out_features: int,
                 activation="linear",
                 temperature=1.0):
        super().__init__()

        self.activation = activation
        self.temperature = temperature
        self.att = nn.Conv1d(
            in_channels=in_features,
            out_channels=out_features,
            kernel_size=1,
            stride=1,
            padding=0,
            bias=True)
        self.cla = nn.Conv1d(
            in_channels=in_features,
            out_channels=out_features,
```

In the `forward` method, it at first calculate self-attention map in the first line `norm_att = torch.softmax(torch.clamp(self.att(x), -10, 10), dim=-1)` . This will be used to aggregate the classification result for segment. In the second line, `cla = self.nonlinear_transform(self.cla(x))` calculates segment wise classification result. Then in the third line, attention aggregation is performed to get clip wise prediction.

Now, let's try to train this model in weakly-supervised manner.

Dataset

In [15]:
```
BIRD_CODE = {
    'aldfly': 0, 'ameavo': 1, 'amebit': 2, 'amecro': 3, 'amegfi': 4,
    'amekes': 5, 'amepip': 6, 'amered': 7, 'amerob': 8, 'amewig': 9,
    'amewoo': 10, 'amtspa': 11, 'annhum': 12, 'astfly': 13, 'baisan': 14,
    'baleag': 15, 'balori': 16, 'banswa': 17, 'barswa': 18, 'bawwar': 19,
    'belkin1': 20, 'belspa2': 21, 'bewwre': 22, 'bkbcuc': 23, 'bkbmag1': 24,
    'bkbwar': 25, 'bkcchi': 26, 'bkchum': 27, 'bkhgro': 28, 'bkpwar': 29,
    'bktspa': 30, 'blkpho': 31, 'blugrb1': 32, 'blujay': 33, 'bnhcow': 34,
    'boboli': 35, 'bongul': 36, 'brdowl': 37, 'brebla': 38, 'brespa': 39,
    'brncre': 40, 'brnthr': 41, 'brthum': 42, 'brwhaw': 43, 'btbwar': 44,
    'btnwar': 45, 'btywar': 46, 'buffle': 47, 'buggna': 48, 'buhvir': 49,
    'bulori': 50, 'bushti': 51, 'buwtea': 52, 'buwwar': 53, 'cacwre': 54,
    'calgul': 55, 'calqua': 56, 'camwar': 57, 'cangoo': 58, 'canwar': 59,
    'canwre': 60, 'carwre': 61, 'casfin': 62, 'caster1': 63, 'casvir': 64,
    'cedwax': 65, 'chispa': 66, 'chiswi': 67, 'chswar': 68, 'chukar': 69,
    'clanut': 70, 'cliswa': 71, 'comgol': 72, 'comgra': 73, 'comloo': 74,
    'commer': 75, 'comnig': 76, 'comrav': 77, 'comred': 78, 'comter': 79,
    'comyel': 80, 'coohaw': 81, 'coshum': 82, 'cowscj1': 83, 'daejun': 84,
    'doccor': 85, 'dowwoo': 86, 'dusfly': 87, 'eargre': 88, 'easblu': 89,
    'easkin': 90, 'easmea': 91, 'easpho': 92, 'eastow': 93, 'eawpew': 94,
    'eucdov': 95, 'eursta': 96, 'evegro': 97, 'fiespa': 98, 'fiscro': 99,
    'foxspa': 100, 'gadwal': 101, 'gcrfin': 102, 'gnttow': 103, 'gnwtea': 10
4,
    'gockin': 105, 'gocspa': 106, 'goleag': 107, 'grbher3': 108, 'grcfly': 1
09,
    'greegr': 110, 'greroa': 111, 'greyel': 112, 'grhowl': 113, 'grnher': 11
4,
    'grtgra': 115, 'grycat': 116, 'gryfly': 117, 'haiwoo': 118, 'hamfly': 11
9,
    'hergul': 120, 'herthr': 121, 'hoomer': 122, 'hoowar': 123, 'horgre': 12
4,
    'horlar': 125, 'houfin': 126, 'houspa': 127, 'houwre': 128, 'indbun': 12
9,
    'juntit1': 130, 'killde': 131, 'labwoo': 132, 'larspa': 133, 'lazbun': 1
34,
    'leabit': 135, 'leafly': 136, 'leasan': 137, 'lecthr': 138, 'lesgol': 13
9,
    'lesnig': 140, 'lesyel': 141, 'lewwoo': 142, 'linspa': 143, 'lobcur': 14
4,
    'lobdow': 145, 'logshr': 146, 'lotduc': 147, 'louwat': 148, 'macwar': 14
9,
    'magwar': 150, 'mallar3': 151, 'marwre': 152, 'merlin': 153, 'moublu': 1
54,
    'mouchi': 155, 'moudov': 156, 'norcar': 157, 'norfli': 158, 'norhar2': 1
59,
    'normoc': 160, 'norpar': 161, 'norpin': 162, 'norsho': 163, 'norwat': 16
4,
    'nrwswa': 165, 'nutwoo': 166, 'olsfly': 167, 'orcwar': 168, 'osprey': 16
9,
    'ovenbi1': 170, 'palwar': 171, 'pasfly': 172, 'pecsan': 173, 'perfal': 1
74,
    'phaino': 175, 'pibgre': 176, 'pilwoo': 177, 'pingro': 178, 'pinjay': 17
9,
    'pinsis': 180, 'pinwar': 181, 'plsvir': 182, 'prawar': 183, 'purfin': 18
4,
    'pygnut': 185, 'rebmer': 186, 'rebnut': 187, 'rebsap': 188, 'rebwoo': 18
9,
    'redcro': 190, 'redhea': 191, 'reevir1': 192, 'renpha': 193, 'reshaw': 1
94,
    'rethaw': 195, 'rewbla': 196, 'ribgul': 197, 'rinduc': 198, 'robgro': 19
9,
    'rocpig': 200, 'rocwre': 201, 'rthhum': 202, 'ruckin': 203, 'rudduc': 20
4,
    'rufgro': 205, 'rufhum': 206, 'rusbla': 207, 'sagspa1': 208, 'sagthr': 2
09,
    'savspa': 210, 'saypho': 211, 'scatan': 212, 'scoori': 213, 'semplo': 21
```

In [16]:
```
PERIOD = 5

class PANNsDataset(data.Dataset):
    def __init__(
            self,
            file_list: List[List[str]],
            waveform_transforms=None):
        self.file_list = file_list  # list of list: [file_path, ebird_code]
        self.waveform_transforms = waveform_transforms

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, idx: int):
        wav_path, ebird_code = self.file_list[idx]

        y, sr = sf.read(wav_path)

        if self.waveform_transforms:
            y = self.waveform_transforms(y)
        else:
            len_y = len(y)
            effective_length = sr * PERIOD
            if len_y < effective_length:
                new_y = np.zeros(effective_length, dtype=y.dtype)
                start = np.random.randint(effective_length - len_y)
                new_y[start:start + len_y] = y
                y = new_y.astype(np.float32)
            elif len_y > effective_length:
                start = np.random.randint(len_y - effective_length)
                y = y[start:start + effective_length].astype(np.float32)
            else:
                y = y.astype(np.float32)

        labels = np.zeros(len(BIRD_CODE), dtype="f")
        labels[BIRD_CODE[ebird_code]] = 1

        return {"waveform": y, "targets": labels}
```

Criterion

In [17]:
```
class PANNsLoss(nn.Module):
    def __init__(self):
        super().__init__()

        self.bce = nn.BCELoss()

    def forward(self, input, target):
        input_ = input["clipwise_output"]
        input_ = torch.where(torch.isnan(input_),
                             torch.zeros_like(input_),
                             input_)
        input_ = torch.where(torch.isinf(input_),
                             torch.zeros_like(input_),
                             input_)

        target = target.float()

        return self.bce(input_, target)
```

Callbacks

In [18]:
```
class F1Callback(Callback):
    def __init__(self,
                 input_key: str = "targets",
                 output_key: str = "logits",
                 model_output_key: str = "clipwise_output",
                 prefix: str = "f1"):
        super().__init__(CallbackOrder.Metric)

        self.input_key = input_key
        self.output_key = output_key
        self.model_output_key = model_output_key
        self.prefix = prefix

    def on_loader_start(self, state: State):
        self.prediction: List[np.ndarray] = []
        self.target: List[np.ndarray] = []

    def on_batch_end(self, state: State):
        targ = state.input[self.input_key].detach().cpu().numpy()
        out = state.output[self.output_key]

        clipwise_output = out[self.model_output_key].detach().cpu().numpy()

        self.prediction.append(clipwise_output)
        self.target.append(targ)

        y_pred = clipwise_output.argmax(axis=1)
        y_true = targ.argmax(axis=1)

        score = f1_score(y_true, y_pred, average="macro")
        state.batch_metrics[self.prefix] = score

    def on_loader_end(self, state: State):
        y_pred = np.concatenate(self.prediction, axis=0).argmax(axis=1)
        y_true = np.concatenate(self.target, axis=0).argmax(axis=1)
        score = f1_score(y_true, y_pred, average="macro")
        state.loader_metrics[self.prefix] = score
        if state.is_valid_loader:
            state.epoch_metrics[state.valid_loader + "_epoch_" +
                                self.prefix] = score
        else:
            state.epoch_metrics["train_epoch_" + self.prefix] = score


class mAPCallback(Callback):
    def __init__(self,
                 input_key: str = "targets",
                 output_key: str = "logits",
                 model_output_key: str = "clipwise_output",
                 prefix: str = "mAP"):
        super().__init__(CallbackOrder.Metric)
        self.input_key = input_key
        self.output_key = output_key
        self.model_output_key = model_output_key
        self.prefix = prefix

    def on_loader_start(self, state: State):
        self.prediction: List[np.ndarray] = []
        self.target: List[np.ndarray] = []

    def on_batch_end(self, state: State):
        targ = state.input[self.input_key].detach().cpu().numpy()
        out = state.output[self.output_key]

        clipwise_output = out[self.model_output_key].detach().cpu().numpy()
```

Train

Some code are taken from https://www.kaggle.com/ttahara/training-birdsong-baseline-resnest50-fast (https://www.kaggle.com/ttahara/training-birdsong-baseline-resnest50-fast) . Thanks @ttahara!

In [19]:
```
tmp_list = []
for audio_d in TRAIN_RESAMPLED_AUDIO_DIRS:
    if not audio_d.exists():
        continue
    for ebird_d in audio_d.iterdir():
        if ebird_d.is_file():
            continue
        for wav_f in ebird_d.iterdir():
            tmp_list.append([ebird_d.name, wav_f.name, wav_f.as_posix()])

train_wav_path_exist = pd.DataFrame(
    tmp_list, columns=["ebird_code", "resampled_filename", "file_path"])

del tmp_list

train_all = pd.merge(
    train, train_wav_path_exist, on=["ebird_code", "resampled_filename"], ho
w="inner")

print(train.shape)
print(train_wav_path_exist.shape)
print(train_all.shape)
```

```
(21375, 38)
(21375, 3)
(21375, 39)
```

In [20]:
```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

train_all["fold"] = -1
for fold_id, (train_index, val_index) in enumerate(skf.split(train_all, trai
n_all["ebird_code"])):
    train_all.iloc[val_index, -1] = fold_id

# # check the propotion
fold_proportion = pd.pivot_table(train_all, index="ebird_code", columns="fol
d", values="xc_id", aggfunc=len)
print(fold_proportion.shape)
```

```
(264, 5)
```

In [21]:
```
use_fold = 0
train_file_list = train_all.query("fold != @use_fold")[["file_path", "ebird_
code"]].values.tolist()
val_file_list = train_all.query("fold == @use_fold")[["file_path", "ebird_co
de"]].values.tolist()

print("[fold {}] train: {}, val: {}".format(use_fold, len(train_file_list),
len(val_file_list)))
```

```
[fold 0] train: 17100, val: 4275
```

In [22]:
```
device = torch.device("cuda:0")

# loaders
loaders = {
    "train": data.DataLoader(PANNsDataset(train_file_list, None),
                             batch_size=64,
                             shuffle=True,
                             num_workers=2,
                             pin_memory=True,
                             drop_last=True),
    "valid": data.DataLoader(PANNsDataset(val_file_list, None),
                             batch_size=64,
                             shuffle=False,
                             num_workers=2,
                             pin_memory=True,
                             drop_last=False)
}

# model
model_config["classes_num"] = 527
model = PANNsCNN14Att(**model_config)
weights = torch.load("../input/pannscnn14-decisionlevelatt-weight/Cnn14_Deci
sionLevelAtt_mAP0.425.pth")
# Fixed in V3
model.load_state_dict(weights["model"])
model.att_block = AttBlock(2048, 264, activation='sigmoid')
model.att_block.init_weights()
model.to(device)

# Optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Scheduler
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)

# Loss
criterion = PANNsLoss().to(device)

# callbacks
callbacks = [
    F1Callback(input_key="targets", output_key="logits", prefix="f1"),
    mAPCallback(input_key="targets", output_key="logits", prefix="mAP"),
    CheckpointCallback(save_n_best=0)
]
```

In [23]:
```
warnings.simplefilter("ignore")

runner = SupervisedRunner(
    device=device,
    input_key="waveform",
    input_target_key="targets")

runner.train(
    model=model,
    criterion=criterion,
    loaders=loaders,
    optimizer=optimizer,
    scheduler=scheduler,
    num_epochs=10,
    verbose=True,
    logdir=f"fold0",
    callbacks=callbacks,
    main_metric="epoch_f1",
    minimize_metric=False)
```

```
1/10 * Epoch (train): 100% 267/267 [03:17<00:00,  1.35it/s, f1=0.000e+00, los
s=0.025, mAP=0.016]
1/10 * Epoch (valid): 100% 67/67 [00:40<00:00,  1.65it/s, f1=0.000e+00, loss=
0.025, mAP=0.004]
[2020-08-14 10:30:58,334]
1/10 * Epoch 1 (_base): lr=0.0010 | momentum=0.9000
1/10 * Epoch 1 (train): epoch_f1=0.0025 | epoch_mAP=0.0043 | f1=0.0019 | loss
=0.0424 | mAP=0.0177
1/10 * Epoch 1 (valid): epoch_f1=0.0004 | epoch_mAP=0.0057 | f1=0.0017 | loss
=0.0253 | mAP=0.0043
2/10 * Epoch (train): 100% 267/267 [03:15<00:00,  1.37it/s, f1=0.000e+00, los
s=0.025, mAP=0.028]
2/10 * Epoch (valid): 100% 67/67 [00:39<00:00,  1.68it/s, f1=0.000e+00, loss=
0.026, mAP=0.006]
[2020-08-14 10:34:53,858]
2/10 * Epoch 2 (_base): lr=0.0008 | momentum=0.9000
2/10 * Epoch 2 (train): epoch_f1=0.0067 | epoch_mAP=0.0073 | f1=0.0060 | loss
=0.0246 | mAP=0.0251
2/10 * Epoch 2 (valid): epoch_f1=0.0096 | epoch_mAP=0.0191 | f1=0.0088 | loss
=0.0249 | mAP=0.0065
3/10 * Epoch (train): 100% 267/267 [03:15<00:00,  1.36it/s, f1=0.057, loss=0.
020, mAP=0.080]
3/10 * Epoch (valid): 100% 67/67 [00:40<00:00,  1.67it/s, f1=0.011, loss=0.02
4, mAP=0.007]
[2020-08-14 10:38:49,616]
3/10 * Epoch 3 (_base): lr=0.0007 | momentum=0.9000
3/10 * Epoch 3 (train): epoch_f1=0.0328 | epoch_mAP=0.0266 | f1=0.0279 | loss
=0.0228 | mAP=0.0532
3/10 * Epoch 3 (valid): epoch_f1=0.0759 | epoch_mAP=0.1140 | f1=0.0232 | loss
=0.0231 | mAP=0.0116
4/10 * Epoch (train): 100% 267/267 [03:14<00:00,  1.37it/s, f1=0.182, loss=0.
017, mAP=0.136]
4/10 * Epoch (valid): 100% 67/67 [00:41<00:00,  1.63it/s, f1=0.054, loss=0.02
0, mAP=0.008]
[2020-08-14 10:42:45,031]
4/10 * Epoch 4 (_base): lr=0.0005 | momentum=0.9000
4/10 * Epoch 4 (train): epoch_f1=0.1339 | epoch_mAP=0.1140 | f1=0.1016 | loss
=0.0190 | mAP=0.1021
4/10 * Epoch 4 (valid): epoch_f1=0.2350 | epoch_mAP=0.3014 | f1=0.0513 | loss
=0.0207 | mAP=0.0149
5/10 * Epoch (train): 100% 267/267 [03:13<00:00,  1.38it/s, f1=0.201, loss=0.
015, mAP=0.152]
5/10 * Epoch (valid): 100% 67/67 [00:41<00:00,  1.63it/s, f1=0.039, loss=0.02
1, mAP=0.009]
[2020-08-14 10:46:39,402]
5/10 * Epoch 5 (_base): lr=0.0004 | momentum=0.9000
5/10 * Epoch 5 (train): epoch_f1=0.2510 | epoch_mAP=0.2368 | f1=0.1838 | loss
=0.0161 | mAP=0.1288
5/10 * Epoch 5 (valid): epoch_f1=0.3548 | epoch_mAP=0.4212 | f1=0.0752 | loss
=0.0178 | mAP=0.0160
6/10 * Epoch (train): 100% 267/267 [03:12<00:00,  1.39it/s, f1=0.312, loss=0.
013, mAP=0.145]
6/10 * Epoch (valid): 100% 67/67 [00:40<00:00,  1.64it/s, f1=0.043, loss=0.02
0, mAP=0.008]
[2020-08-14 10:50:32,456]
6/10 * Epoch 6 (_base): lr=0.0002 | momentum=0.9000
6/10 * Epoch 6 (train): epoch_f1=0.3333 | epoch_mAP=0.3216 | f1=0.2422 | loss
=0.0146 | mAP=0.1404
6/10 * Epoch 6 (valid): epoch_f1=0.4094 | epoch_mAP=0.4496 | f1=0.0853 | loss
=0.0173 | mAP=0.0161
7/10 * Epoch (train): 100% 267/267 [03:10<00:00,  1.40it/s, f1=0.333, loss=0.
012, mAP=0.159]
7/10 * Epoch (valid): 100% 67/67 [00:41<00:00,  1.62it/s, f1=0.066, loss=0.01
9, mAP=0.009]
[2020-08-14 10:54:24,379]
7/10 * Epoch 7 (_base): lr=0.0001 | momentum=0.9000
```

Seems it's learning something.

Now I'll show how this model works in the inference phase. I'll use trained model of this which I trained by myself using the data of this competition in my local environment.

Since several concerns (https://www.kaggle.com/c/birdsong-recognition/discussion/172356) are expressed about over-sharing of top solutions during competition, and since I do respect those people who have worked hard to improve their scores, I would not make trained weight in common and would not share how I trained this model.

Prediction with SED model

In [24]:
```
model_config = {
    "sample_rate": 32000,
    "window_size": 1024,
    "hop_size": 320,
    "mel_bins": 64,
    "fmin": 50,
    "fmax": 14000,
    "classes_num": 264
}

weights_path = "../input/birdcall-pannsatt-aux-weak/best.pth"
```

In [25]:
```
def get_model(config: dict, weights_path: str):
    model = PANNsCNN14Att(**config)
    checkpoint = torch.load(weights_path)
    model.load_state_dict(checkpoint["model_state_dict"])
    device = torch.device("cuda")
    model.to(device)
    model.eval()
    return model
```

In [26]:
```
def prediction_for_clip(test_df: pd.DataFrame,
                        clip: np.ndarray,
                        model: PANNsCNN14Att,
                        threshold=0.5):
    PERIOD = 30
    audios = []
    y = clip.astype(np.float32)
    len_y = len(y)
    start = 0
    end = PERIOD * SR
    while True:
        y_batch = y[start:end].astype(np.float32)
        if len(y_batch) != PERIOD * SR:
            y_pad = np.zeros(PERIOD * SR, dtype=np.float32)
            y_pad[:len(y_batch)] = y_batch
            audios.append(y_pad)
            break
        start = end
        end += PERIOD * SR
        audios.append(y_batch)

    array = np.asarray(audios)
    tensors = torch.from_numpy(array)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model.eval()
    estimated_event_list = []
    global_time = 0.0
    site = test_df["site"].values[0]
    audio_id = test_df["audio_id"].values[0]
    for image in progress_bar(tensors):
        image = image.view(1, image.size(0))
        image = image.to(device)

        with torch.no_grad():
            prediction = model(image)
            framewise_outputs = prediction["framewise_output"].detach(
                ).cpu().numpy()[0]

        thresholded = framewise_outputs >= threshold

        for target_idx in range(thresholded.shape[1]):
            if thresholded[:, target_idx].mean() == 0:
                pass
            else:
                detected = np.argwhere(thresholded[:, target_idx]).reshape(-
1)
                head_idx = 0
                tail_idx = 0
                while True:
                    if (tail_idx + 1 == len(detected)) or (
                            detected[tail_idx + 1] -
                            detected[tail_idx] != 1):
                        onset = 0.01 * detected[
                            head_idx] + global_time
                        offset = 0.01 * detected[
                            tail_idx] + global_time
                        onset_idx = detected[head_idx]
                        offset_idx = detected[tail_idx]
                        max_confidence = framewise_outputs[
                            onset_idx:offset_idx, target_idx].max()
                        mean_confidence = framewise_outputs[
                            onset_idx:offset_idx, target_idx].mean()
                        estimated_event = {
                            "site": site,
                            "audio_id": audio_id,
```

In [27]:
```
def prediction(test_df: pd.DataFrame,
               test_audio: Path,
               model_config: dict,
               weights_path: str,
               threshold=0.5):
    model = get_model(model_config, weights_path)
    unique_audio_id = test_df.audio_id.unique()

    warnings.filterwarnings("ignore")
    prediction_dfs = []
    for audio_id in unique_audio_id:
        with timer(f"Loading {audio_id}"):
            clip, _ = librosa.load(test_audio / (audio_id + ".mp3"),
                                   sr=SR,
                                   mono=True,
                                   res_type="kaiser_fast")

        test_df_for_audio_id = test_df.query(
            f"audio_id == '{audio_id}'").reset_index(drop=True)
        with timer(f"Prediction on {audio_id}"):
            prediction_df = prediction_for_clip(test_df_for_audio_id,
                                                clip=clip,
                                                model=model,
                                                threshold=threshold)

        prediction_dfs.append(prediction_df)

    prediction_df = pd.concat(prediction_dfs, axis=0, sort=False).reset_inde
x(drop=True)
    return prediction_df
```

In [28]:
```
prediction_df = prediction(test_df=test,
                           test_audio=TEST_AUDIO_DIR,
                           model_config=model_config,
                           weights_path=weights_path,
                           threshold=0.5)
```

```
[Loading 41e6fe6504a34bf6846938ba78d13df1] start
[Loading 41e6fe6504a34bf6846938ba78d13df1] done in 2.31 s
[Prediction on 41e6fe6504a34bf6846938ba78d13df1] start
```

100.00% [1/1 00:00<00:00]

```
[Prediction on 41e6fe6504a34bf6846938ba78d13df1] done in 0.60 s
[Loading cce64fffafed40f2b2f3d3413ec1c4c2] start
[Loading cce64fffafed40f2b2f3d3413ec1c4c2] done in 0.81 s
[Prediction on cce64fffafed40f2b2f3d3413ec1c4c2] start
```

100.00% [2/2 00:00<00:00]

```
[Prediction on cce64fffafed40f2b2f3d3413ec1c4c2] done in 0.07 s
[Loading 99af324c881246949408c0b1ae54271f] start
[Loading 99af324c881246949408c0b1ae54271f] done in 0.82 s
[Prediction on 99af324c881246949408c0b1ae54271f] start
```

100.00% [2/2 00:00<00:00]

```
[Prediction on 99af324c881246949408c0b1ae54271f] done in 0.08 s
[Loading 6ab74e177aa149468a39ca10beed6222] start
[Loading 6ab74e177aa149468a39ca10beed6222] done in 0.76 s
[Prediction on 6ab74e177aa149468a39ca10beed6222] start
```

100.00% [2/2 00:00<00:00]

```
[Prediction on 6ab74e177aa149468a39ca10beed6222] done in 0.08 s
[Loading b2fd3f01e9284293a1e33f9c811a2ed6] start
[Loading b2fd3f01e9284293a1e33f9c811a2ed6] done in 0.78 s
[Prediction on b2fd3f01e9284293a1e33f9c811a2ed6] start
```

100.00% [2/2 00:00<00:00]

```
[Prediction on b2fd3f01e9284293a1e33f9c811a2ed6] done in 0.07 s
[Loading de62b37ebba749d2abf29d4a493ea5d4] start
[Loading de62b37ebba749d2abf29d4a493ea5d4] done in 0.44 s
[Prediction on de62b37ebba749d2abf29d4a493ea5d4] start
```

100.00% [1/1 00:00<00:00]

```
[Prediction on de62b37ebba749d2abf29d4a493ea5d4] done in 0.04 s
[Loading 8680a8dd845d40f296246dbed0d37394] start
[Loading 8680a8dd845d40f296246dbed0d37394] done in 0.88 s
[Prediction on 8680a8dd845d40f296246dbed0d37394] start
```

100.00% [2/2 00:00<00:00]

```
[Prediction on 8680a8dd845d40f296246dbed0d37394] done in 0.07 s
[Loading 940d546e5eb745c9a74bce3f35efa1f9] start
[Loading 940d546e5eb745c9a74bce3f35efa1f9] done in 1.22 s
[Prediction on 940d546e5eb745c9a74bce3f35efa1f9] start
```

100.00% [3/3 00:00<00:00]

```
[Prediction on 940d546e5eb745c9a74bce3f35efa1f9] done in 0.11 s
[Loading 07ab324c602e4afab65ddbcc746c31b5] start
[Loading 07ab324c602e4afab65ddbcc746c31b5] done in 0.66 s
[Prediction on 07ab324c602e4afab65ddbcc746c31b5] start
```

100.00% [1/1 00:00<00:00]

```
[Prediction on 07ab324c602e4afab65ddbcc746c31b5] done in 0.04 s
[Loading 899616723a32409c996f6f3441646c2a] start
[Loading 899616723a32409c996f6f3441646c2a] done in 1.17 s
[Prediction on 899616723a32409c996f6f3441646c2a] start
```

100.00% [2/2 00:00<00:00]

```
[Prediction on 899616723a32409c996f6f3441646c2a] done in 0.08 s
[Loading 9cc5d9646f344f1bbb52640a988fe902] start
[Loading 9cc5d9646f344f1bbb52640a988fe902] done in 3.42 s
[Prediction on 9cc5d9646f344f1bbb52640a988fe902] start
```

100.00% [9/9 00:00<00:00]

```
[Prediction on 9cc5d9646f344f1bbb52640a988fe902] done in 0.32 s
[Loading a56e20a518684688a9952add8a9d5213] start
[Loading a56e20a518684688a9952add8a9d5213] done in 0.74 s
[Prediction on a56e20a518684688a9952add8a9d5213] start
```

100.00% [2/2 00:00<00:00]

```
[Prediction on a56e20a518684688a9952add8a9d5213] done in 0.08 s
[Loading 967798362887457283069 03d54e264dd] start
[Loading 967798362887457283069 03d54e264dd] done in 0.59 s
[Prediction on 967798362887457283069 03d54e264dd] start
```

100.00% [1/1 00:00<00:00]

```
[Prediction on 967798362887457283069 03d54e264dd] done in 0.04 s
[Loading f77783ba4c6641bc918b034a18c23e53] start
[Loading f77783ba4c6641bc918b034a18c23e53] done in 0.47 s
[Prediction on f77783ba4c6641bc918b034a18c23e53] start
```

100.00% [1/1 00:00<00:00]

```
[Prediction on f77783ba4c6641bc918b034a18c23e53] done in 0.04 s
[Loading 856b194b097441958697c2bcd1f63982] start
[Loading 856b194b097441958697c2bcd1f63982] done in 0.71 s
[Prediction on 856b194b097441958697c2bcd1f63982] start
```

100.00% [1/1 00:00<00:00]

```
[Prediction on 856b194b097441958697c2bcd1f63982] done in 0.04 s
```

In [29]:
```
prediction_df
```

Out[29]:

	site	audio_id	ebird_code	onset	offset	max_confidence	mean_confider
0	site_1	41e6fe6504a34bf6846938ba78d13df1	aldfly	0.96	2.23	0.985395	0.897(
1	site_1	41e6fe6504a34bf6846938ba78d13df1	aldfly	7.04	7.67	0.526611	0.519
2	site_1	41e6fe6504a34bf6846938ba78d13df1	aldfly	11.20	12.15	0.956318	0.928
3	site_1	41e6fe6504a34bf6846938ba78d13df1	aldfly	14.40	15.03	0.809643	0.805
4	site_1	41e6fe6504a34bf6846938ba78d13df1	aldfly	20.16	21.43	0.987058	0.917(
...	...	...	...	...	...	...	...
195	site_3	856b194b097441958697c2bcd1f63982	aldfly	18.24	19.19	0.885321	0.789(
196	site_3	856b194b097441958697c2bcd1f63982	aldfly	19.84	22.07	0.983291	0.913(
197	site_3	856b194b097441958697c2bcd1f63982	aldfly	23.04	23.67	0.669237	0.624
198	site_3	856b194b097441958697c2bcd1f63982	aldfly	25.92	26.87	0.950051	0.897
199	site_3	856b194b097441958697c2bcd1f63982	aldfly	27.84	28.79	0.991087	0.899

200 rows × 7 columns

Postprocess

In [30]:
```
labels = {}

for audio_id, sub_df in prediction_df.groupby("audio_id"):
    events = sub_df[["ebird_code", "onset", "offset", "max_confidence", "sit
e"]].values
    n_events = len(events)
    removed_event = []
    # Overlap deletion: this part may not be necessary
    # I deleted this part in other model and found there's no difference on
the public LB score.
    for i in range(n_events):
        for j in range(n_events):
            if i == j:
                continue
            if i in removed_event:
                continue
            if j in removed_event:
                continue

            event_i = events[i]
            event_j = events[j]

            if (event_i[1] - event_j[2] >= 0) or (event_j[1] - event_i[2] >=
0):
                pass
            else:
                later_onset = max(event_i[1], event_j[1])
                sooner_onset = min(event_i[1], event_j[1])
                sooner_offset = min(event_i[2], event_j[2])
                later_offset = max(event_i[2], event_j[2])

                intersection = sooner_offset - later_onset
                union = later_offset - sooner_onset

                iou = intersection / union
                if iou > 0.4:
                    if event_i[3] > event_j[3]:
                        removed_event.append(j)
                    else:
                        removed_event.append(i)

    site = events[0][4]
    for i in range(n_events):
        if i in removed_event:
            continue
        event = events[i][0]
        onset = events[i][1]
        offset = events[i][2]
        if site in {"site_1", "site_2"}:
            start_section = int((onset // 5) * 5) + 5
            end_section = int((offset // 5) * 5) + 5
            cur_section = start_section

            row_id = f"{site}_{audio_id}_{start_section}"
            if labels.get(row_id) is not None:
                labels[row_id].add(event)
            else:
                labels[row_id] = set()
                labels[row_id].add(event)

            while cur_section != end_section:
                cur_section += 5
                row_id = f"{site}_{audio_id}_{cur_section}"
                if labels.get(row_id) is not None:
                    labels[row_id].add(event)
                else:
```

In [31]:
```
for key in labels:
    labels[key] = " ".join(sorted(list(labels[key])))


row_ids = list(labels.keys())
birds = list(labels.values())
post_processed = pd.DataFrame({
    "row_id": row_ids,
    "birds": birds
})
post_processed.head()
```

Out[31]:

	row_id	birds
0	site_2_07ab324c602e4afab65ddbcc746c31b5_5	aldfly
1	site_2_07ab324c602e4afab65ddbcc746c31b5_10	aldfly
2	site_2_07ab324c602e4afab65ddbcc746c31b5_15	aldfly
3	site_2_07ab324c602e4afab65ddbcc746c31b5_25	redcro
4	site_1_41e6fe6504a34bf6846938ba78d13df1_5	aldfly

In [32]:
```
all_row_id = test[["row_id"]]
submission = all_row_id.merge(post_processed, on="row_id", how="left")
submission = submission.fillna("nocall")
submission.to_csv("submission.csv", index=False)
submission.head(20)
```

Out[32]:

	row_id	birds
0	site_1_41e6fe6504a34bf6846938ba78d13df1_5	aldfly
1	site_1_41e6fe6504a34bf6846938ba78d13df1_10	aldfly fiespa
2	site_1_41e6fe6504a34bf6846938ba78d13df1_15	aldfly moudov
3	site_1_41e6fe6504a34bf6846938ba78d13df1_20	aldfly chswar
4	site_1_41e6fe6504a34bf6846938ba78d13df1_25	aldfly
5	site_1_cce64fffafed40f2b2f3d3413ec1c4c2_5	aldfly
6	site_1_cce64fffafed40f2b2f3d3413ec1c4c2_10	nocall
7	site_1_cce64fffafed40f2b2f3d3413ec1c4c2_15	aldfly
8	site_1_cce64fffafed40f2b2f3d3413ec1c4c2_20	nocall
9	site_1_cce64fffafed40f2b2f3d3413ec1c4c2_25	nocall
10	site_1_cce64fffafed40f2b2f3d3413ec1c4c2_30	nocall
11	site_1_cce64fffafed40f2b2f3d3413ec1c4c2_35	aldfly
12	site_1_99af324c881246949408c0b1ae54271f_5	hamfly
13	site_1_99af324c881246949408c0b1ae54271f_10	aldfly
14	site_1_99af324c881246949408c0b1ae54271f_15	aldfly
15	site_1_99af324c881246949408c0b1ae54271f_20	aldfly
16	site_1_99af324c881246949408c0b1ae54271f_25	aldfly
17	site_1_99af324c881246949408c0b1ae54271f_30	aldfly
18	site_1_99af324c881246949408c0b1ae54271f_35	aldfly
19	site_1_6ab74e177aa149468a39ca10beed6222_5	aldfly

EOF

In [ ]: