Update

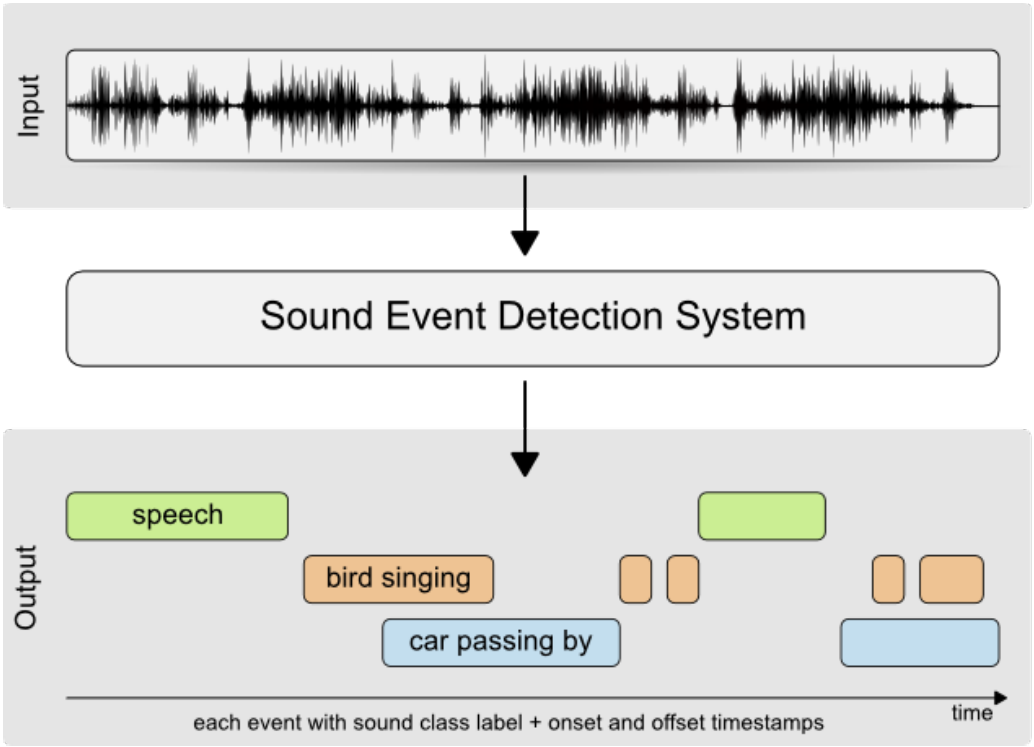
Version note

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),
```

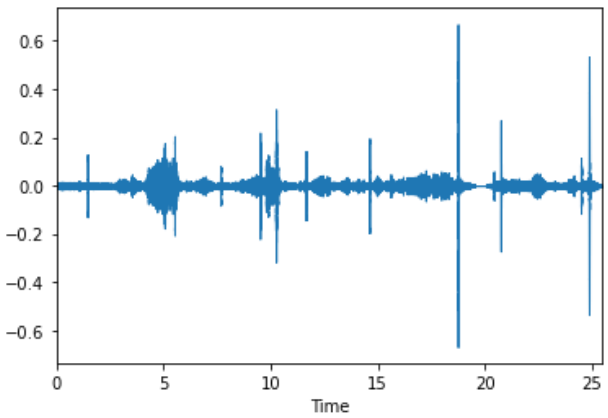
In [8]:
```
class PANNsCNN14Att(nn.Module):
    def __init__(self, sample_rate: int, window_size: int, hop_size: int,
                 mel_bins: int, fmin: int, fmax: int, classes_num: int):
        super().__init__()

        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);
```



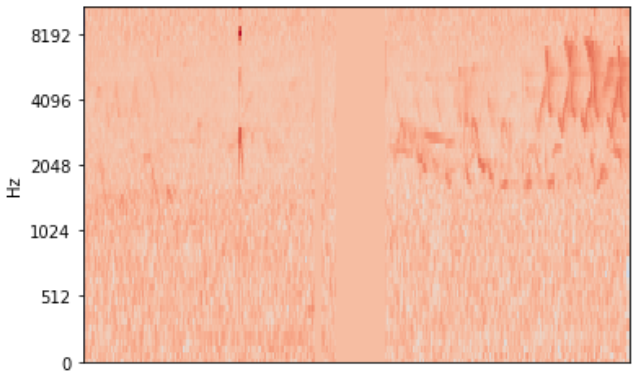
In [11]:
```
model_config = {
    "sample_rate": 32000,
    "window_size": 1024,
    "hop_size": 320,
    "mel_bins": 64,
    "fmin": 50,
    "fmax": 14000,
    "classes_num": 264
}

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

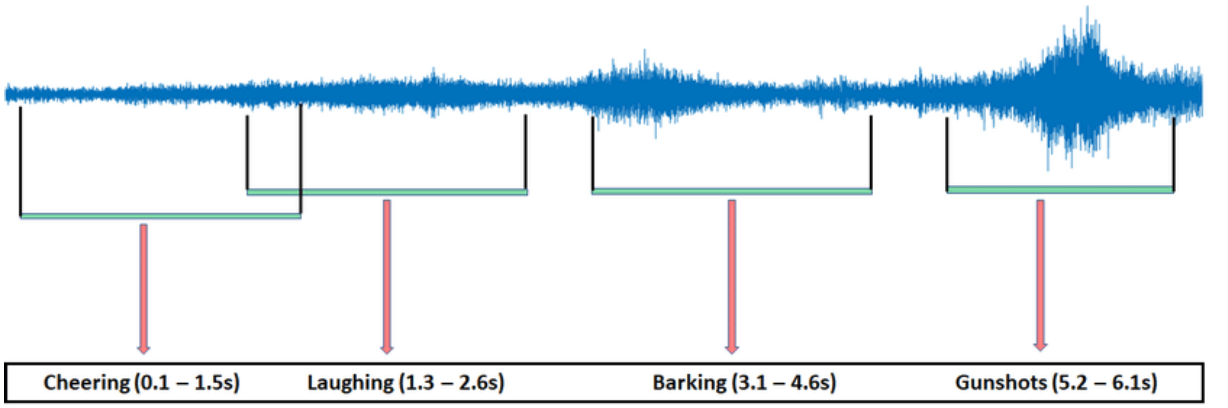


Illustration of Strong Labeling of Events

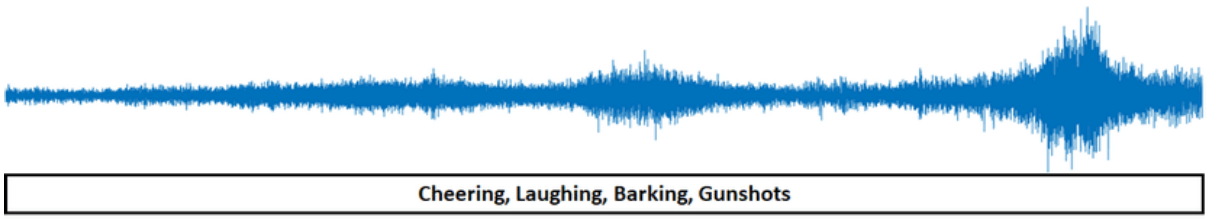


Illustration of Weak Labeling of Events

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 [ ]:

All cridets @hidehisaarai1213 (https://www.kaggle.com/hidehisaarai1213)

This notebook based on this Introduction to Sound Event Detection (https://www.kaggle.com/hidehisaarai1213/introduction-to-sound-event-detection)

Install packages

In [1]:
```
!pip -q install --upgrade pip
!pip -q install timm
!pip -q install torchlibrosa
!pip -q install audiomentations
```

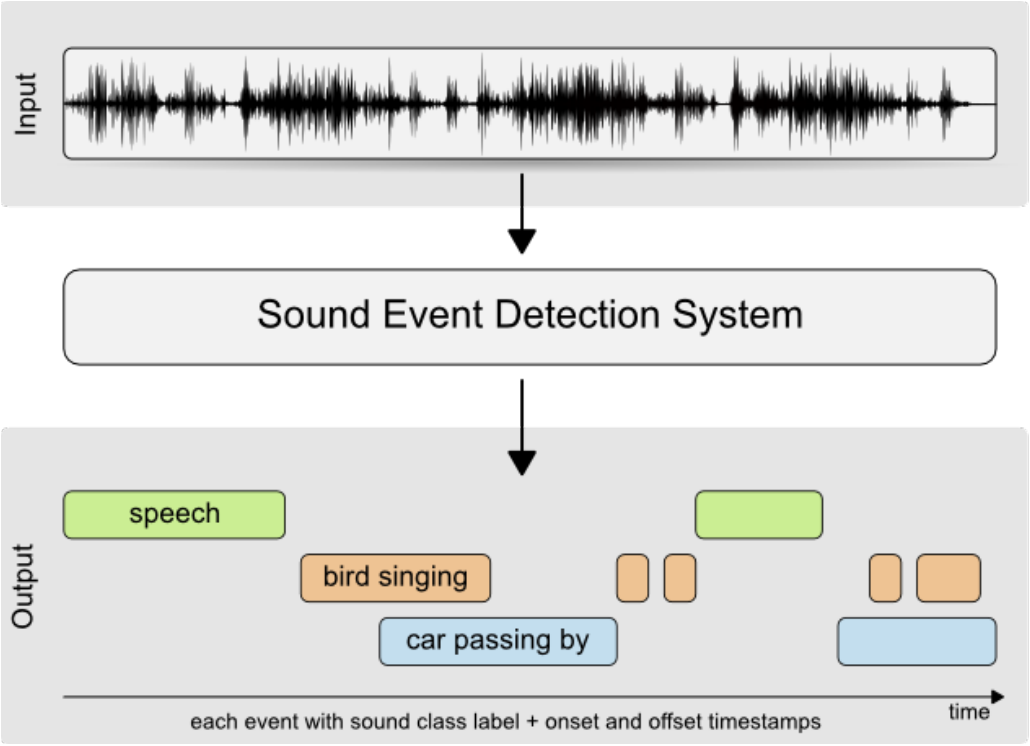
import packages

In [2]:
```
import os, glob, random, time
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
import librosa, librosa.display
import soundfile as sf

import torch
import torch.nn as nn
import torch.nn.functional as F

from tqdm import tqdm
from functools import partial
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from transformers import get_linear_schedule_with_warmup
from torchlibrosa.stft import Spectrogram, LogmelFilterBank
from torchlibrosa.augmentation import SpecAugmentation

import timm
from timm.models.efficientnet import tf_efficientnet_b0_ns
```

About Sound Event Detection(SED)

Sound event detection (SED) is the task of detecting the type as well as the onset and offset times of sound events in audio streams.

In this notebook i will show how to train Sound Event Detection (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.

for more details

-> Polyphonic Sound Event Detection with Weak Labeling Paper (http://www.cs.cmu.edu/~yunwang/papers/cmu-thesis.pdf)

-> Introduction to Sound Event Detection Notebook (https://www.kaggle.com/hidehisaarai1213/introduction-to-sound-event-detection)

PANN Utils

-> PANNs repository (https://github.com/qiuqiangkong/audioset_tagging_cnn/)

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

In [3]:
```
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 init_weights(model):
    classname = model.__class__.__name__
    if classname.find("Conv2d") != -1:
        nn.init.xavier_uniform_(model.weight, gain=np.sqrt(2))
        model.bias.data.fill_(0)
    elif classname.find("BatchNorm") != -1:
        model.weight.data.normal_(1.0, 0.02)
        model.bias.data.fill_(0)
    elif classname.find("GRU") != -1:
        for weight in model.parameters():
            if len(weight.size()) > 1:
                nn.init.orghogonal_(weight.data)
    elif classname.find("Linear") != -1:
        model.weight.data.normal_(0, 0.01)
        model.bias.data.zero_()

def do_mixup(x: torch.Tensor, mixup_lambda: torch.Tensor):
    """Mixup x of even indexes (0, 2, 4, ...) with x of odd indexes
    (1, 3, 5, ...).
    Args:
      x: (batch_size * 2, ...)
      mixup_lambda: (batch_size * 2,)
    Returns:
      out: (batch_size, ...)
    """
    out = (x[0::2].transpose(0, -1) * mixup_lambda[0::2] +
           x[1::2].transpose(0, -1) * mixup_lambda[1::2]).transpose(0, -1)
    return out


class Mixup(object):
    def __init__(self, mixup_alpha, random_seed=1234):
        """Mixup coefficient generator.
        """
        self.mixup_alpha = mixup_alpha
        self.random_state = np.random.RandomState(random_seed)

    def get_lambda(self, batch_size):
        """Get mixup random coefficients.
        Args:
          batch_size: int
        Returns:
          mixup_lambdas: (batch_size,)
        """
        mixup_lambdas = []
        for n in range(0, batch_size, 2):
            lam = self.random_state.beta(self.mixup_alpha, self.mixup_alpha,
1)[0]
            mixup_lambdas.append(lam)
            mixup_lambdas.append(1. - lam)

        return torch.from_numpy(np.array(mixup_lambdas, dtype=np.float32))
```

Create Folds

In [4]:
```
FOLDS = 5
SEED = 42

train = pd.read_csv("../input/rfcx-species-audio-detection/train_tp.csv").so
rt_values("recording_id")
ss = pd.read_csv("../input/rfcx-species-audio-detection/sample_submission.cs
v")

train_gby = train.groupby("recording_id")[["species_id"]].first().reset_inde
x()
train_gby = train_gby.sample(frac=1, random_state=SEED).reset_index(drop=Tru
e)
train_gby.loc[:, 'kfold'] = -1

X = train_gby["recording_id"].values
y = train_gby["species_id"].values

kfold = StratifiedKFold(n_splits=FOLDS)
for fold, (t_idx, v_idx) in enumerate(kfold.split(X, y)):
    train_gby.loc[v_idx, "kfold"] = fold

train = train.merge(train_gby[['recording_id', 'kfold']], on="recording_id",
how="left")
print(train.kfold.value_counts())
train.to_csv("train_folds.csv", index=False)
```

```
3    249
2    246
4    243
0    241
1    237
Name: kfold, dtype: int64
```

SED Model

1. Model takes raw waveform and converted into log-melspectogram using `torchlibrosa` 's module
2. spectogram converted into 3-channels input for ImageNet pretrain model to extract features from CNN's
3. Although it's downsized through several convolution and pooling layers, the size of it's third dimension and it still contains time information. Each element of this dimension is segment. In SED model, we provide prediction for each of this.

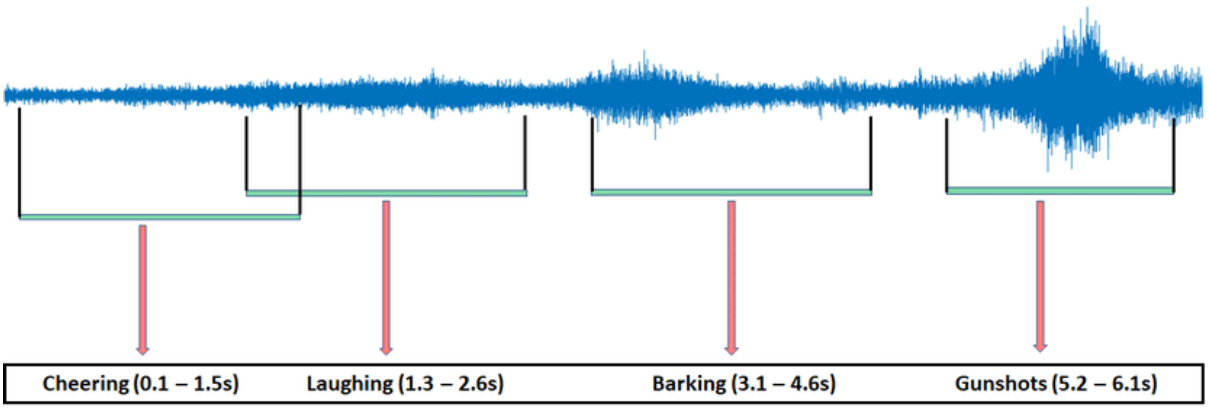


Illustration of Strong Labeling of Events

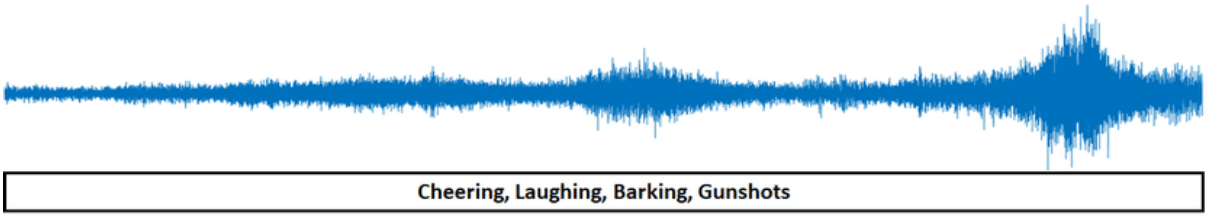


Illustration of Weak Labeling of Events

1. 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.
2. 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.

In [5]:
```
encoder_params = {
    "tf_efficientnet_b0_ns": {
        "features": 1280,
        "init_op": partial(tf_efficientnet_b0_ns, pretrained=True, drop_path
_rate=0.2)
    }
}


class AudioSEDModel(nn.Module):
    def __init__(self, encoder, sample_rate, window_size, hop_size, mel_bins
, fmin, fmax, classes_num):
        super().__init__()

        window = 'hann'
        center = True
        pad_mode = 'reflect'
        ref = 1.0
        amin = 1e-10
        top_db = None
        self.interpolate_ratio = 30  # Downsampled ratio

        # Spectrogram extractor
        self.spectrogram_extractor = Spectrogram(n_fft=window_size, hop_leng
th=hop_size,
            win_length=window_size, window=window, center=center, pad_mode=p
ad_mode,
            freeze_parameters=True)

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

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

        # Model Encoder
        self.encoder = encoder_params[encoder]["init_op"]()
        self.fc1 = nn.Linear(encoder_params[encoder]["features"], 1024, bias
=True)
        self.att_block = AttBlock(1024, classes_num, activation="sigmoid")
        self.bn0 = nn.BatchNorm2d(mel_bins)
        self.init_weight()

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

    def forward(self, input, mixup_lambda=None):
        """Input : (batch_size, data_length)"""

        x = self.spectrogram_extractor(input)
        # batch_size x 1 x time_steps x freq_bins
        x = self.logmel_extractor(x)
        # batch_size x 1 x time_steps x mel_bins

        frames_num = x.shape[2]

        x = x.transpose(1, 3)
        x = self.bn0(x)
        x = x.transpose(1, 3)
```

Dataset

In [6]:
```
def crop_or_pad(y, sr, period, record, mode="train"):
    len_y = len(y)
    effective_length = sr * period
    rint = np.random.randint(len(record['t_min']))
    time_start = record['t_min'][rint] * sr
    time_end = record['t_max'][rint] * sr
    if len_y > effective_length:
        # Positioning sound slice
        center = np.round((time_start + time_end) / 2)
        beginning = center - effective_length / 2
        if beginning < 0:
            beginning = 0
        beginning = np.random.randint(beginning, center)
        ending = beginning + effective_length
        if ending > len_y:
            ending = len_y
        beginning = ending - effective_length
        y = y[beginning:ending].astype(np.float32)
    else:
        y = y.astype(np.float32)
        beginning = 0
        ending = effective_length


    beginning_time = beginning / sr
    ending_time = ending / sr
    label = np.zeros(24, dtype='f')

    for i in range(len(record['t_min'])):
        if (record['t_min'][i] <= ending_time) and (record['t_max'][i] >= be
ginning_time):
            label[record['species_id'][i]] = 1

    return y, label
```

In [7]:
```
class SedDataset:
    def __init__(self, df, period=10, stride=5, audio_transform=None, data_p
ath="train", mode="train"):

        self.period = period
        self.stride = stride
        self.audio_transform = audio_transform
        self.data_path = data_path
        self.mode = mode

        self.df = df.groupby("recording_id").agg(lambda x: list(x)).reset_in
dex()

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

    def __getitem__(self, idx):
        record = self.df.iloc[idx]

        y, sr = sf.read(f"{self.data_path}/{record['recording_id']}.flac")

        if self.mode != "test":
            y, label = crop_or_pad(y, sr, period=self.period, record=record,
mode=self.mode)

            if self.audio_transform:
                y = self.audio_transform(samples=y, sample_rate=sr)
        else:
            y_ = []
            i = 0
            effective_length = self.period * sr
            stride = self.stride * sr
            y = np.stack([y[i:i+effective_length].astype(np.float32) for i i
n range(0, 60*sr+stride-effective_length, stride)])
            label = np.zeros(24, dtype='f')
            if self.mode == "valid":
                for i in record['species_id']:
                    label[i] = 1

        return {
            "image" : y,
            "target" : label,
            "id" : record['recording_id']
        }
```

Augmentations

In [8]:
```
import audiomentations as AA

train_audio_transform = AA.Compose([
    AA.AddGaussianNoise(p=0.5),
    AA.AddGaussianSNR(p=0.5),
    #AA.AddBackgroundNoise("../input/train_audio/", p=1)
    #AA.AddImpulseResponse(p=0.1),
    #AA.AddShortNoises("../input/train_audio/", p=1)
    #AA.FrequencyMask(min_frequency_band=0.0,  max_frequency_band=0.2, p=0.1
),
    #AA.TimeMask(min_band_part=0.0, max_band_part=0.2, p=0.1),
    #AA.PitchShift(min_semitones=-0.5, max_semitones=0.5, p=0.1),
    #AA.Shift(p=0.1),
    #AA.Normalize(p=0.1),
    #AA.ClippingDistortion(min_percentile_threshold=0, max_percentile_thresh
old=1, p=0.05),
    #AA.PolarityInversion(p=0.05),
    #AA.Gain(p=0.2)
])
```

Utils

In [9]:
```
def _lwlrap_sklearn(truth, scores):
    """Reference implementation from https://colab.research.google.com/drive
/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8"""
    sample_weight = np.sum(truth > 0, axis=1)
    nonzero_weight_sample_indices = np.flatnonzero(sample_weight > 0)
    overall_lwlrap = metrics.label_ranking_average_precision_score(
        truth[nonzero_weight_sample_indices, :] > 0,
        scores[nonzero_weight_sample_indices, :],
        sample_weight=sample_weight[nonzero_weight_sample_indices])
    return overall_lwlrap

class AverageMeter(object):
    """Computes and stores the average and current value"""

    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count

class MetricMeter(object):
    def __init__(self):
        self.reset()

    def reset(self):
        self.y_true = []
        self.y_pred = []

    def update(self, y_true, y_pred):
        self.y_true.extend(y_true.cpu().detach().numpy().tolist())
        self.y_pred.extend(y_pred.cpu().detach().numpy().tolist())

    @property
    def avg(self):
        #score_class, weight = lwlrap(np.array(self.y_true), np.array(self.y
_pred))
        self.score = _lwlrap_sklearn(np.array(self.y_true), np.array(self.y_
pred)) #(score_class * weight).sum()
        return {
            "lwlrap" : self.score
        }

def seed_everithing(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
```

Losses

In [10]:
```
from torch.nn import BCEWithLogitsLoss, CrossEntropyLoss

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)
```

Functions

In [11]:
```
def train_epoch(args, model, loader, criterion, optimizer, scheduler, epoch)
:
    losses = AverageMeter()
    scores = MetricMeter()

    model.train()
    t = tqdm(loader)
    for i, sample in enumerate(t):
        optimizer.zero_grad()
        input = sample['image'].to(args.device)
        target = sample['target'].to(args.device)
        output = model(input)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if scheduler and args.step_scheduler:
            scheduler.step()

        bs = input.size(0)
        scores.update(target, torch.sigmoid(torch.max(output['framewise_outp
ut'], dim=1)[0]))
        losses.update(loss.item(), bs)

        t.set_description(f"Train E:{epoch} - Loss{losses.avg:0.4f}")
    t.close()
    return scores.avg, losses.avg

def valid_epoch(args, model, loader, criterion, epoch):
    losses = AverageMeter()
    scores = MetricMeter()
    model.eval()
    with torch.no_grad():
        t = tqdm(loader)
        for i, sample in enumerate(t):
            input = sample['image'].to(args.device)
            target = sample['target'].to(args.device)
            output = model(input)
            loss = criterion(output, target)

            bs = input.size(0)
            scores.update(target, torch.sigmoid(torch.max(output['framewise_
output'], dim=1)[0]))
            losses.update(loss.item(), bs)
            t.set_description(f"Valid E:{epoch} - Loss:{losses.avg:0.4f}")
    t.close()
    return scores.avg, losses.avg

def test_epoch(args, model, loader):
    model.eval()
    pred_list = []
    id_list = []
    with torch.no_grad():
        t = tqdm(loader)
        for i, sample in enumerate(t):
            input = sample["image"].to(args.device)
            bs, seq, w = input.shape
            input = input.reshape(bs*seq, w)
            id = sample["id"]
            output = model(input)
            output = torch.sigmoid(torch.max(output['framewise_output'], dim
=1)[0])
            output = output.reshape(bs, seq, -1)
            output = torch.sum(output, dim=1)
            #output, _ = torch.max(output, dim=1)
            output = output.cpu().detach().numpy().tolist()
            pred_list.extend(output)
```

Main Function

In [12]:
```
def main(fold):
    seed_everithing(args.seed)

    args.fold = fold
    args.save_path = os.path.join(args.output_dir, args.exp_name)
    os.makedirs(args.save_path, exist_ok=True)

    train_df = pd.read_csv(args.train_csv)
    sub_df = pd.read_csv(args.sub_csv)
    if args.DEBUG:
        train_df = train_df.sample(200)
    train_fold = train_df[train_df.kfold != fold]
    valid_fold = train_df[train_df.kfold == fold]

    train_dataset = SedDataset(
        df = train_fold,
        period=args.period,
        audio_transform=train_audio_transform,
        data_path=args.train_data_path,
        mode="train"
    )

    valid_dataset = SedDataset(
        df = valid_fold,
        period=args.period,
        stride=5,
        audio_transform=None,
        data_path=args.train_data_path,
        mode="valid"
    )

    test_dataset = SedDataset(
        df = sub_df,
        period=args.period,
        stride=5,
        audio_transform=None,
        data_path=args.test_data_path,
        mode="test"
    )

    train_loader = torch.utils.data.DataLoader(
        train_dataset,
        batch_size=args.batch_size,
        shuffle=True,
        drop_last=True,
        num_workers=args.num_workers
    )

    valid_loader = torch.utils.data.DataLoader(
        valid_dataset,
        batch_size=args.batch_size,
        shuffle=False,
        drop_last=False,
        num_workers=args.num_workers
    )

    test_loader = torch.utils.data.DataLoader(
        test_dataset,
        batch_size=args.batch_size,
        shuffle=False,
        drop_last=False,
        num_workers=args.num_workers
    )

    model = AudioSEDModel(**args.model_param)
    model = model.to(args.device)
```

Config

In [13]:
```
class args:
    DEBUG = False

    exp_name = "SED_E0_5F_BASE"
    pretrain_weights = None
    model_param = {
        'encoder' : 'tf_efficientnet_b0_ns',
        'sample_rate': 48000,
        'window_size' : 512, #* 2, # 512 * 2
        'hop_size' : 512, #345 * 2, # 320
        'mel_bins' : 128, # 60
        'fmin' : 0,
        'fmax' : 48000 // 2,
        'classes_num' : 24
    }
    period = 10
    seed = 42
    start_epcoh = 0
    epochs = 50
    lr = 1e-3
    batch_size = 16
    num_workers = 4
    early_stop = 15
    step_scheduler = True
    epoch_scheduler = False

    device = ('cuda' if torch.cuda.is_available() else 'cpu')
    train_csv = "train_folds.csv"
    test_csv = "test_df.csv"
    sub_csv = "../input/rfcx-species-audio-detection/sample_submission.csv"
    output_dir = "weights"
    train_data_path = "../input/rfcx-species-audio-detection/train"
    test_data_path = "../input/rfcx-species-audio-detection/test"
```

train folds

In [14]:
```
main(fold=0)
```

```
/opt/conda/lib/python3.7/site-packages/librosa/filters.py:239: UserWarning: E
mpty filters detected in mel frequency basis. Some channels will produce empt
y responses. Try increasing your sampling rate (and fmax) or reducing n_mels.
  "Empty filters detected in mel frequency basis. "
Downloading: "https://github.com/rwightman/pytorch-image-models/releases/down
load/v0.1-weights/tf_efficientnet_b0_ns-c0e6a31c.pth" to /root/.cache/torch/h
ub/checkpoints/tf_efficientnet_b0_ns-c0e6a31c.pth
Train E:0 - Loss0.4852: 100%|██████████| 56/56 [00:55<00:00,  1.01it/s]
Valid E:0 - Loss:0.4059: 100%|██████████| 15/15 [00:09<00:00,  1.56it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 01:56:28 2021

               Fold:0, Epoch:0, lr:0.0002

               Train Loss:0.4852 - LWLRAP:0.1627

               Valid Loss:0.4059 - LWLRAP:0.1606


########## >>>>>>>> Model Improved From -inf ----> 0.16058852440783872

Train E:1 - Loss0.1944: 100%|██████████| 56/56 [00:51<00:00,  1.09it/s]
Valid E:1 - Loss:0.1987: 100%|██████████| 15/15 [00:08<00:00,  1.72it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 01:57:28 2021

               Fold:0, Epoch:1, lr:0.0004

               Train Loss:0.1944 - LWLRAP:0.1730

               Valid Loss:0.1987 - LWLRAP:0.1861


########## >>>>>>>> Model Improved From 0.16058852440783872 ----> 0.186063221
59243144

Train E:2 - Loss0.1820: 100%|██████████| 56/56 [00:50<00:00,  1.11it/s]
Valid E:2 - Loss:0.1793: 100%|██████████| 15/15 [00:09<00:00,  1.63it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 01:58:28 2021

               Fold:0, Epoch:2, lr:0.0006

               Train Loss:0.1820 - LWLRAP:0.2071

               Valid Loss:0.1793 - LWLRAP:0.2616


########## >>>>>>>> Model Improved From 0.18606322159243144 ----> 0.261594997
39783953

Train E:3 - Loss0.1677: 100%|██████████| 56/56 [00:50<00:00,  1.10it/s]
Valid E:3 - Loss:0.1680: 100%|██████████| 15/15 [00:08<00:00,  1.72it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 01:59:27 2021

               Fold:0, Epoch:3, lr:0.0008

               Train Loss:0.1677 - LWLRAP:0.2382

               Valid Loss:0.1680 - LWLRAP:0.2615
```

```
Train E:4 - Loss0.1626: 100%|██████████| 56/56 [00:50<00:00,  1.11it/s]
Valid E:4 - Loss:0.1596: 100%|██████████| 15/15 [00:08<00:00,  1.73it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 02:00:27 2021

               Fold:0, Epoch:4, lr:0.001

               Train Loss:0.1626 - LWLRAP:0.2791

               Valid Loss:0.1596 - LWLRAP:0.3310


########## >>>>>>>> Model Improved From 0.26159499739783953 ----> 0.331033178
5228701

Train E:5 - Loss0.1545: 100%|██████████| 56/56 [00:50<00:00,  1.10it/s]
Valid E:5 - Loss:0.1447: 100%|██████████| 15/15 [00:08<00:00,  1.75it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 02:01:27 2021

               Fold:0, Epoch:5, lr:0.0009777778

               Train Loss:0.1545 - LWLRAP:0.3401

               Valid Loss:0.1447 - LWLRAP:0.4175


########## >>>>>>>> Model Improved From 0.3310331785228701 ----> 0.4175091080
802941

Train E:6 - Loss0.1469: 100%|██████████| 56/56 [00:51<00:00,  1.09it/s]
Valid E:6 - Loss:0.1438: 100%|██████████| 15/15 [00:08<00:00,  1.75it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 02:02:27 2021

               Fold:0, Epoch:6, lr:0.0009555556

               Train Loss:0.1469 - LWLRAP:0.3933

               Valid Loss:0.1438 - LWLRAP:0.4622


########## >>>>>>>> Model Improved From 0.4175091080802941 ----> 0.4621947390
481543

Train E:7 - Loss0.1384: 100%|██████████| 56/56 [00:50<00:00,  1.11it/s]
Valid E:7 - Loss:0.1338: 100%|██████████| 15/15 [00:09<00:00,  1.61it/s]
  0%|          | 0/56 [00:00<?, ?it/s]

               Thu Jan 14 02:03:27 2021

               Fold:0, Epoch:7, lr:0.0009333333

               Train Loss:0.1384 - LWLRAP:0.4226

               Valid Loss:0.1338 - LWLRAP:0.5219


########## >>>>>>>> Model Improved From 0.4621947390481543 ----> 0.5218585147
926521

Train E:8 - Loss0.1326: 100%|██████████| 56/56 [00:50<00:00,  1.10it/s]
Valid E:8 - Loss:0.1223: 100%|██████████| 15/15 [00:09<00:00,  1.57it/s]
  0%|          | 0/56 [00:00<?, ?it/s]
```

PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition

Qiuqiang Kong, *Student Member, IEEE*, Yin Cao, *Member, IEEE*, Turab Iqbal,
Yuxuan Wang, Wenwu Wang, *Senior Member, IEEE* and Mark D. Plumbley, *Fellow, IEEE*

***Abstract*—Audio pattern recognition is an important research topic in the machine learning area, and includes several tasks such as audio tagging, acoustic scene classification, music classification, speech emotion classification and sound event detection. Recently, neural networks have been applied to tackle audio pattern recognition problems. However, previous systems are built on specific datasets with limited durations. Recently, in computer vision and natural language processing, systems pretrained on large-scale datasets have generalized well to several tasks. However, there is limited research on pretraining systems on large-scale datasets for audio pattern recognition. In this paper, we propose pretrained audio neural networks (PANNs) trained on the large-scale AudioSet dataset. These PANNs are transferred to other audio related tasks. We investigate the performance and computational complexity of PANNs modeled by a variety of convolutional neural networks. We propose an architecture called Wavegram-Logmel-CNN using both log-mel spectrogram and waveform as input feature. Our best PANN system achieves a state-of-the-art mean average precision (mAP) of 0.439 on AudioSet tagging, outperforming the best previous system of 0.392. We transfer PANNs to six audio pattern recognition tasks, and demonstrate state-of-the-art performance in several of those tasks. We have released the source code and pretrained models of PANNs: https://github.com/qiuqiangkong/audioset_tagging_cnn.**

***Index Terms*—Audio tagging, pretrained audio neural networks, transfer learning.**

I. Introduction

Audio pattern recognition is an important research topic in the machine learning area, and plays an important role in our life. We are surrounded by sounds that contain rich information of where we are, and what events are happening around us. Audio pattern recognition contains several tasks such as audio tagging [1], acoustic scene classification [2], music classification [3], speech emotion classification and sound event detection [4].

Audio pattern recognition has attracted increasing research interest in recent years. Early audio pattern recognition work focused on private datasets collected by individual researchers [5][6]. For example, Woodard [5] applied a hidden Markov model (HMM) to classify three types of sounds: wooden door open and shut, dropped metal and poured water. Recently, the Detection and Classification of Acoustic Scenes and Events (DCASE) challenge series [7][8][9][2] have provided publicly available datasets, such as acoustic scene classification and sound event detection datasets. The DCASE challenges have attracted increasing research interest in audio pattern recognition. For example, the recent DCASE 2019 challenge received 311 entries across five subtasks [10].

However, it is still an open question how well an audio pattern recognition system can perform when trained on large-scale datasets. In computer vision, several image classification systems have been built with the large-scale ImageNet dataset [11]. In natural language processing, several language models have been built with the large-scale text datasets such as Wikipedia [12]. However, systems trained on large-scale audio datasets have been more limited [1][13][14][15].

A milestone for audio pattern recognition was the release of AudioSet [1], a dataset containing over 5,000 hours of audio recordings with 527 sound classes. Instead of releasing the raw audio recordings, AudioSet released embedding features of audio clips extracted from a pretrained convolutional neural network [13]. Several researchers have investigated building systems with those embedding features [13][16][17][18][19][20]. However, the embedding features may not be an optimal representation for audio recordings, which may limit the performance of those systems. In this article, we propose pretrained audio neural networks (PANNs) trained on raw AudioSet recordings with a wide range of neural networks. We show that several PANN systems outperform previous state-of-the-art audio tagging systems. We also investigate the audio tagging performance and computation complexities of PANNs.

We propose that PANNs can be transferred to other audio pattern recognition tasks. Previous researchers have previously investigated transfer learning for audio tagging. For example, audio tagging systems were pretrained on the Million Song Dataset were proposed in [21], with embedding features extracted from pretrained convolutional neural networks (CNNs) are used as inputs to second-stage classifiers such as neural networks or support vector machines (SVMs) [14][22]. Systems pretrained on MagnaTagATune [23] and acoustic scene [24] datasets were fine-tuned on other audio tagging tasks [25][26]. These transfer learning systems were mainly trained with music datasets, and were limited to smaller datasets than AudioSet.

The contribution of this work includes: (1) We introduce

Q. Kong, Y. Cao, T. Iqbal, and M. D. Plumbley are with the Centre for Vision, Speech and Signal Processing, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: q.kong@surrey.ac.uk; yin.cao@surrey.ac.uk; t.iqbal@surrey.ac.uk; m.plumbley@surrey.ac.uk).

This work was supported in part by the EPSRC Grant EP/N014111/1 "Making Sense of Sounds", in part by the Research Scholarship from the China Scholarship Council 201406150082, and in part by a studentship (Reference: 1976218) from the EPSRC Doctoral Training Partnership under Grant EP/N509772/1. This work was supported by National Natural Science Foundation of China (Grant No. 11804365). *(Qiuqiang Kong is first author.) (Yin Cao is corresponding author.)*

Y. Wang is with the ByteDance AI Lab, Mountain View, CA, USA (e-mail: wangyuxuan.11@bytedance.com).

W. Wang is with the Centre for Vision, Speech and Signal Processing, University of Surrey, Guildford GU2 7XH, U.K., and also with Qingdao University of Science and Technology, Qingdao 266071, China (e-mail: w.wang@surrey.ac.uk).

PANNs trained on AudioSet with 1.9 million audio clips with an ontology of 527 sound classes; (2) We investigate the trade-off between audio tagging performance and computation complexity of a wide range of PANNs; (3) We propose a system that we call Wavegram-Logmel-CNN that achieves a mean average precision (mAP) of 0.439 on AudioSet tagging, outperforming previous state-of-the-art system with an mAP 0.392 and Google's system with an mAP 0.314; (4) We show that PANNs can be transferred to other audio pattern recognition tasks, outperforming several state-of-the-art systems; (5) We have released the source code and pretrained PANN models.

This paper is organized as follows: Section II introduces audio tagging with various convolutional neural networks; Section III introduces our proposed Wavegram-CNN systems; Section IV introduces our data processing techniques, including data balancing and data augmentation for AudioSet tagging; Section VI shows experimental results, and Section VII concludes this work.

TABLE I
CNNs FOR AUDIOSET TAGGING

VGGish [1]	CNN6	CNN10	CNN14
Log-mel spectrogram 96 frames × 64 mel bins	Log-mel spectrogram 1000 frames × 64 mel bins		
3×3 @ 64 ReLU	5×5 @ 64 BN, ReLU	$\begin{pmatrix} 3 \times 3 \text{ @ } 64 \\ \text{BN, ReLU} \end{pmatrix} \times 2$	$\begin{pmatrix} 3 \times 3 \text{ @ } 64 \\ \text{BN, ReLU} \end{pmatrix} \times 2$
MP 2×2	Pooling 2×2		
3×3 @ 128 ReLU	5×5 @ 128 BN, ReLU	$\begin{pmatrix} 3 \times 3 \text{ @ } 128 \\ \text{BN, ReLU} \end{pmatrix} \times 2$	$\begin{pmatrix} 3 \times 3 \text{ @ } 128 \\ \text{BN, ReLU} \end{pmatrix} \times 2$
MP 2×2	Pooling 2×2		
$\begin{pmatrix} 3 \times 3 \text{ @ } 256 \\ \text{ReLU} \end{pmatrix} \times 2$	5×5 @ 256 BN, ReLU	$\begin{pmatrix} 3 \times 3 \text{ @ } 256 \\ \text{BN, ReLU} \end{pmatrix} \times 2$	$\begin{pmatrix} 3 \times 3 \text{ @ } 256 \\ \text{BN, ReLU} \end{pmatrix} \times 2$
MP 2×2	Pooling 2×2		
$\begin{pmatrix} 3 \times 3 \text{ @ } 512 \\ \text{ReLU} \end{pmatrix} \times 2$	5×5 @ 512 BN, ReLU	$\begin{pmatrix} 3 \times 3 \text{ @ } 512 \\ \text{BN, ReLU} \end{pmatrix} \times 2$	$\begin{pmatrix} 3 \times 3 \text{ @ } 512 \\ \text{BN, ReLU} \end{pmatrix} \times 2$
MP 2×2 Flatten	Global pooling		Pooling 2×2
FC 4096 ReLU $\times 2$	FC 512, ReLU		$\begin{pmatrix} 3 \times 3 \text{ @ } 1024 \\ \text{BN, ReLU} \end{pmatrix} \times 2$
FC 527, Sigmoid	FC 527, Sigmoid		Pooling 2×2
			$\begin{pmatrix} 3 \times 3 \text{ @ } 2048 \\ \text{BN, ReLU} \end{pmatrix} \times 2$
			Global pooling
			FC 2048, ReLU
			FC 527, Sigmoid

II. AUDIO TAGGING SYSTEMS

Audio tagging is an essential task of audio pattern recognition, with the goal of predicting the presence or absence of audio tags in an audio clip. Early work on audio tagging included using manually-designed features as input, such as audio energy, zero-crossing rate, and mel-frequency cepstrum coefficients (MFCCs) [27]. Generative models, including Gaussian mixture models (GMMs) [28][29], hidden Markov models (HMMs), and discriminative support vector machines (SVMs) [30] have been used as classifiers. Recently, neural network based methods such as convolutional neural networks (CNNs) have been used [3] to predict the tags of audio recordings. CNN-based systems have achieved state-of-the-art performance in several DCASE challenge tasks including acoustic scene classification [2] and sound event detection [4]. However, many of those works focused on particular tasks with a limited number of sound classes, and were not designed to recognize a wide range of sound classes. In this article, we focus on training large-scale PANNs on AudioSet [1] to tackle the general audio tagging problem.

A. CNNs

1) Conventional CNNs: CNNs have been successfully applied to computer vision tasks such as image classification [31][32]. A CNN consists of several convolutional layers. Each convolutional layer contains several kernels that are convolved with the input feature maps to capture their local patterns. CNNs adopted for audio tagging [3][20] often use log mel spectrograms as input [3][20]. Short time Fourier transforms (STFTs) are applied to time-domain waveforms to calculate spectrograms. Then, mel filter banks are applied to the spectrograms, followed by a logarithmic operation to extract log mel spectrograms [3][20].

2) Adapting CNNs for AudioSet tagging: The PANNs we use are based on our previously-proposed cross-task CNN systems for the DCASE 2019 challenge [33], with an extra fully-connected layer added to the penultimate layer of CNNs to further increase the representation ability. We investigate 6-, 10- and 14-layer CNNs. The 6-layer CNN consists of 4 convolutional layers with a kernel size of 5×5, based on AlexNet [34]. The 10- and 14-layer CNNs consist of 4 and 6 convolutional layers, respectively, inspired by the VGG-like CNNs [35]. Each convolutional block consists of 2 convolutional layers with a kernel size of 3×3. Batch normalization [36] is applied between each convolutional layer, and the ReLU nonlinearity [37] is used to speed up and stabilize the training. We apply average pooling of size of 2×2 to each convolutional block for downsampling, as 2×2 average pooling has been shown to outperform 2×2 max pooling [33].

Global pooling is applied after the last convolutional layer to summarize the feature maps into a fixed-length vector. In [15], maximum and average operation were used for global pooling. To combine their advantages, we sum the averaged and maximized vectors. In our previous work [33], those fixed-length vectors were used as embedding features for audio clips. In this work, we add an extra fully-connected layer to the fixed length vectors to extract embedding features which can further increase their representation ability. For a particular audio pattern recognition task, a linear classifier is applied to the embedding features, followed by either a softmax nonlinearity for classification tasks or a sigmoid nonlinearity for tagging tasks. Dropout [38] is applied after each downsampling operation and fully connected layers to prevent systems from overfitting. Table I summarizes our proposed CNN systems. The number after the "@" symbol indicates the number of feature maps. The first column shows the VGGish network proposed by [13]. MP is the abbreviation of max pooling. The "Pooling 2×2" in Table I is average pooling with size of 2×2. In [13], an audio clip was split into 1-second segments, [13] also assumed each segment inherits the label of the audio clip, which may lead to incorrect labels. In contrast, our systems from the second to the fourth columns in Table I applies an entire audio clip for training without cutting the audio clip into segments.

TABLE II
RESNETS FOR AUDIOSET TAGGING

ResNet22	ResNet38	ResNet54
Log mel spectrogram 1000 frames ×64 mel bins		
$\left(3 \times 3 \text{ @ } 512, \text{BN}, \text{ReLU}\right) \times 2$		
Pooling 2×2		
$\left(\text{BasicB @ } 64\right) \times 2$	$\left(\text{BasicB @ } 64\right) \times 3$	$\left(\text{BottleneckB @ } 64\right) \times 3$
Pooling 2×2		
$\left(\text{BasicB @ } 128\right) \times 2$	$\left(\text{BasicB @ } 128\right) \times 4$	$\left(\text{BottleneckB @ } 128\right) \times 4$
Pooling 2×2		
$\left(\text{BasicB @ } 256\right) \times 2$	$\left(\text{BasicB @ } 256\right) \times 6$	$\left(\text{BottleneckB @ } 256\right) \times 6$
Pooling 2×2		
$\left(\text{BasicB @ } 512\right) \times 2$	$\left(\text{BasicB @ } 512\right) \times 3$	$\left(\text{BottleneckB @ } 512\right) \times 3$
Pooling 2×2		
$\left(3 \times 3 \text{ @ } 2048, \text{BN}, \text{ReLU}\right) \times 2$		
Global pooling		
FC 2048, ReLU		
FC 527, Sigmoid		

We denote the waveform of an audio clip as x_n, where n is the index of audio clips, and $f(x_n) \in [0,1]^K$ is the output of a PANN representing the presence probabilities of K sound classes. The label of x_n is denoted as $y_n \in \{0,1\}^K$. A binary cross-entropy loss function l is used to train a PANN:

$$l = -\sum_{n=1}^{N}(y_n \cdot \ln f(x_n) + (1-y_n) \cdot \ln(1-f(x_n)), \qquad (1)$$

where N is the number of training clips in AudioSet. In training, the parameters of $f(\cdot)$ are optimized by using gradient descent methods to minimize the loss function l.

B. ResNets

1) Conventional residual networks (ResNets): Deeper CNNs have been shown to achieve better performance than shallower CNNs for audio classification [31]. One challenge of very deep conventional CNNs is that the gradients do not propagate properly from the top layers to the bottom layers [32]. To address this problem, ResNets [32] introduced shortcut connections between convolutional layers. In this way, the forward and backward signals can be propagated from one layer to any other layer directly. The shortcut connections only introduce a small number of extra parameters and a little additional computational complexity. A ResNet consists of several blocks, where each block consists of two convolutional layers with a kernel size of 3×3, and a shortcut connection between input and output. Each bottleneck block consists of three convolutional layers with a network-in-network architecture [39] that can be used instead of the basic blocks in a ResNet [32].

2) Adapting ResNets for AudioSet tagging: We adapt ResNet [32] for AudioSet tagging as follows. To begin with, two convolutional layers and a downsampling layer are applied on the log mel spectrogram to reduce the input log mel spectrogram size. We implement three types of ResNets with different depths: a 22-layer ResNet with 8 basic blocks; a 38-layer ResNet with 16 basic blocks, and a 54-layer ResNet with 16 residual blocks. Table II shows the architecture of the ResNet systems adapted for AudioSet tagging. The BasicB and BottleneckB are abbreviations of basic block and bottleneck block, respectively.

TABLE III
MOBILENETS FOR AUDIOSET TAGGING

MobileNetV1	MobileNetV2
3×3 @ 32, BN, ReLU	
Pooling 2×2	
V1Block @ 64 V1Block @ 128	V2Block, t=1 @ 16 (V2Block, t=6 @ 24) × 2
Pooling 2×2	
V1Block @ 128 V1Block @ 256	(V2Block, t=6 @ 32) × 3
Pooling 2×2	
V1Block @ 256 V1Block @ 512	(V2Block, t=6 @ 64) × 4
Pooling 2×2	
(V1Block @ 512) × 5 V1Block @ 1024	(V2Block, t=6 @ 96) × 3
Pooling 2×2	
V1Block @ 1024	(V2Block, t=6 @ 160) × 3 (V2Block, t=6 @ 320) × 1
Global pooling	
FC, 1024, ReLU	
FC, 527, Sigmoid	

C. MobileNets

1) Conventional MobileNets: Computational complexity is an important issue when systems are implemented on portable devices. Compared to CNNs and ResNets, MobileNets were intended to reduce the number of parameters and multiply-add operations in a CNN. MobileNets were based on depthwise separable convolutions [40] by factorizing a standard convolution into a depthwise convolution and a 1×1 pointwise convolution [40].

2) Adapting MobileNets for AudioSet tagging: We adapt MobileNetV1 [40] and MobileNetV2 [41] systems for AudioSet tagging shown in Table III. The V1Blocks and V2Blocks are MobileNet convolutional blocks [40][41], each consisting of two and three convolutional layers, respectively.

D. One-dimensional CNNs

Previous audio tagging systems were based on the log mel spectrogram, a hand-crafted feature. To improve performance, several researchers proposed to build one-dimensional CNNs which operate directly on the time-domain waveforms. For example, Dai et al. [31] proposed a one-dimensional CNN for acoustic scene classification, and Lee et al. [42] proposed a one-dimensional CNN that was later adopted by Pons et al. [15] for music tagging.

1) DaiNet: DaiNet [31] applied kernels of length 80 and stride 4 to the input waveform of audio recordings. The kernels are learnable during training. To begin with, a maximum operation is applied to the first convolutional layer, which is designed to make the system be robust to phase shift of the input signals. Then, several one-dimensional convolutional blocks with kernel size 3 and stride 4 were applied to extract high level features. An 18-layer DaiNet with four convolutional layers in each convolutional block achieved the best result in UrbanSound8K [43] classification [31].

2) LeeNet: In contrast to DaiNet that applied large kernels in the first layer, LeeNet [42] applied small kernels with length 3 on the waveforms, to replace the STFT for spectrogram

extraction. LeeNet consists of several one dimensional convolutional layers, each followed by a downsampling layer of size 2. The original LeeNet consists of 11 layers.

3) Adapting one-dimensional CNNs for AudioSet tagging: We modify LeeNet by extending it to a deeper architecture with 24 layers, replacing each convolutional layer with a convolutional block that consists of two convolutional layers. To further increase the number of layers of the one-dimensional CNNs, we propose a one-dimensional residual network (Res1dNet) with a small kernel size of 3. We replace the convolutional blocks in LeeNet with residual blocks, where each residual block consists of two convolutional layers with kernel size 3. The first and second convolutional layers of convolutional block have dilations of 1 and 2, respectively, to increase the receptive field of the corresponding residual block. Downsampling is applied after each residual block. By using 14 and 24 residual blocks, we obtain a Res1dNet31 and a Res1dNet51 with 31 and 51 layers, respectively.

III. Wavegram-CNN systems

Previous one-dimensional CNN systems [31][42][15] have not outperformed the systems trained with log mel spectrograms as input. One characteristic of previous time-domain CNN systems [31][42] is that they were not designed to capture frequency information, because there is no frequency axis in the one-dimensional CNN systems, so they can not capture frequency patterns of a sound event with different pitch shifts.

A. Wavegram-CNN systems

In this section, we propose architectures which we call Wavegram-CNN and Wavegram-Logmel-CNN for AudioSet tagging. The Wavegram-CNN we propose is a time-domain audio tagging system. Wavegram is a feature we propose that is similar to log mel spectrogram, but is learnt using a neural network. A Wavegram is designed to learn a time-frequency representation that is a modification of the Fourier transform. A Wavegram has a time axis and a frequency axis. Frequency patterns are important for audio pattern recognition, for example, sounds with different pitch shifts belong to the same class. A Wavegram is designed to learn frequency information that may be lacking in one-dimensional CNN systems. Wavegrams may also improve over hand-crafted log mel spectrograms by learning a new kind of time-frequency transform from data. Wavegrams can then replace log mel spectrograms as input features resulting in our Wavegram-CNN system. We also combine the Wavegram and the log mel spectrogram as a new feature to build the Wavegram-Logmel-CNN system as shown in Fig. 1.

To build a Wavegram, we first apply a one-dimensional CNN to time-domain waveform. The one-dimensional CNN begins with a convolutional layer with filter length 11 and stride 5 to reduce the size of the input. This immediately reduces the input lengths by a factor of 5 times to reduce memory usage. This is followed by three convolutional blocks, where each convolutional block consists of two convolutional layers with dilations of 1 and 2, respectively, which are designed to increase the receptive field of the convolutional layers. Each convolutional block is followed by a downsampling layer with stride 4. By using the stride and downsampling three times, we downsample a 32 kHz audio recording to $32,000/5/4/4/4 = 100$ frames of features per second. We denote the output size of the one-dimensional CNN layers as $T \times C$, where T is the number of frames and C is the number of channels. We reshape this output to a tensor with a size of $T \times F \times C/F$ by splitting C channels into C/F groups, where each group has F frequency bins. We call this tensor a *Wavegram*. The Wavegram learns frequency information by introducing F frequency bins in each of C/F channels. We apply CNN14 described in Section II-A as a backbone architecture on the extracted Wavegram, so that we can fairly compare the Wavegram and log mel spectrogram based systems. Two dimensional CNNs such as CNN14 can capture time-frequency invariant patterns on the Wavegram, because kernels are convolved along both time and frequency axis in a Wavegram.

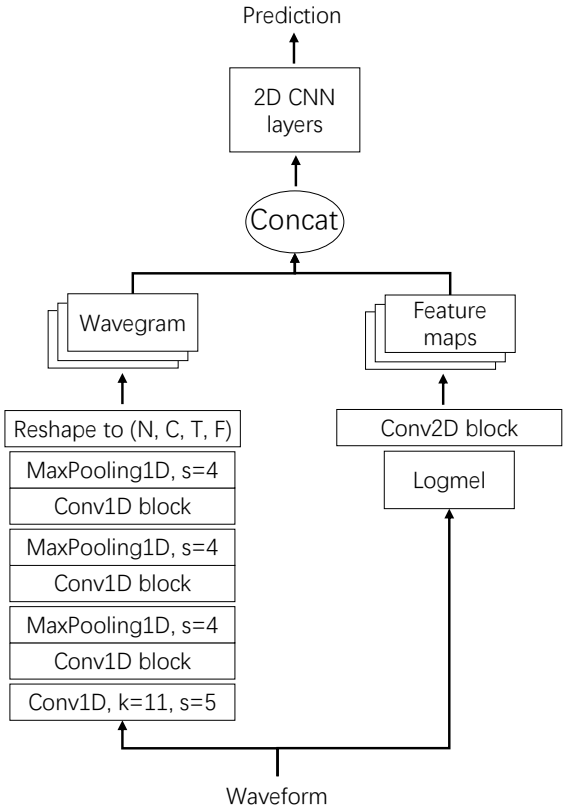


Fig. 1. Architecture of Wavegram-Logmel-CNN

B. Wavegram-Logmel-CNN

Furthermore, we can combine the Wavegram and log mel spectrogram into a new representation. In this way, we can utilize the information from both time-domain waveforms and log mel spectrograms. The combination is carried out along the channel dimension. The Wavegram provides extra information for audio tagging, complementing the log mel spectrogram. Fig. 1 shows the architecture of the Wavegram-Logmel-CNN.

IV. Data processing

In this section, we introduce data processing for AudioSet tagging, including data balancing and data augmentation. Data balancing is a technique used to train neural networks on a highly unbalanced dataset. Data augmentation is a technique used to augment the dataset, to prevent systems from overfitting during training.

A. Data balancing

The number of audio clips available for training varies from sound class to sound class. For example, there are over 900,000 audio clips belonging to the categories "Speech" and "Music". On the other hand, there are only tens of audio clips belonging to the category "Toothbrush". The number of audio clips of different sound classes has a long tailed distribution. Training data are input to a PANN in mini-batches during training. Without a data balancing strategy, audio clips are uniformly sampled from AudioSet. Therefore, sound classes with more training clips such as "Speech" are more likely to be sampled during training. In an extreme case, all data in a mini-batch may belong to the same sound class. This will cause the PANN to overfit to sound classes with more training clips, and underfit to sound classes with fewer training clips. To solve this problem, we design a balanced sampling strategy to train PANNs. That is, audio clips are approximately equally sampled from all sound classes to constitute a mini-batch. We use the term "approximately" because an audio clip may contain more than one tag.

B. Data augmentation

Data augmentation is a useful way to prevent a system from overfitting. Some sound classes in AudioSet contain only a small number (e.g., hundreds) of training clips which may limit the performance of PANNs. We apply mixup [44] and SpecAugment [45] to augment data during training.

1) Mixup: Mixup [44] is a way to augment a dataset by interpolating both the input and target of two audio clips from a dataset. For example, we denote the input of two audio clips as x_1, x_2, and their targets as y_1, y_2, respectively. Then, the augmented input and target can be obtained by $x = \lambda x_1 + (1-\lambda)x_2$ and $y = \lambda y_1 + (1-\lambda)y_2$ respectively, where λ is sampled from a Beta distribution [44]. By default, we apply mixup on the log mel spectrogram. We will compare the performance of mixup augmentation on the log mel spectrogram and on the time-domain waveform in Section VI-C4.

2) SpecAugment: SpecAugment [45] was proposed for augmenting speech data for speech recognition. SpecAugment operates on the log mel spectrogram of an audio clip using frequency masking and time masking. Frequency masking is applied such that f consecutive mel frequency bins $[f_0, f_0 + f]$ are masked, where f is chosen from a uniform distribution from 0 to a frequency mask parameter f', and f_0 is chosen from $[0, F - f]$, where F is the number of mel frequency bins [45]. There can be more than one frequency mask in each log mel spectrogram. The frequency mask can improve the robustness of PANNs to frequency distortion of audio clips [45]. Time masking is similar to frequency masking, but is applied in the time domain.

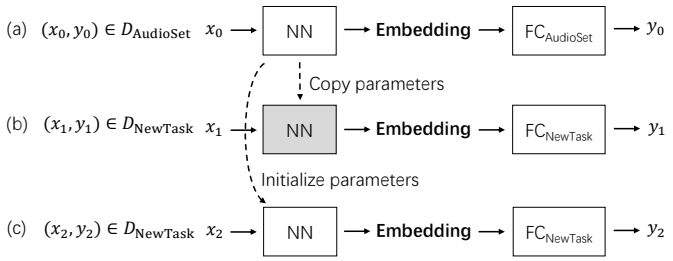


Fig. 2. (a) A PANN is pretrained with the AudioSet dataset. (b) For a new task, the PANN is used as a feature extractor. A classifier is built on the extracted embedding features. The shaded rectangle indicates the parameters are frozen and not trained. (c) For a new task, the parameters of a neural network are initialized with a PANN. Then, all parameters are fine-tuned on the new task.

V. Transfer to other tasks

To investigate the generalization ability of PANNs, we transfer PANNs to a range of audio pattern recognition tasks. Previous works on audio transfer learning [21][14][22][25][23] mainly focused on music tagging, and were limited to smaller datasets than AudioSet. To begin with, we demonstrate the training of a PANN in Fig. 2(a). Here, D_{AudioSet} is the AudioSet dataset, and x_0, y_0 are training input and target, respectively. $\text{FC}_{\text{AudioSet}}$ is the fully connected layer for AudioSet tagging. In this article, we propose to compare the following transfer learning strategies.

1) Train a system from scratch. All parameters are randomly initialized. Systems are similar to PANNs, except for the final fully-connected layer which depends on the task dependent number of outputs. This system is used as a baseline system to be compared with other transfer learning systems.

2) Use a PANN as a feature extractor. For new tasks, the embedding features of audio clips are calculated by using the PANN. Then, the embedding features are used as input to a classifier, such as a fully-connected neural network. When training on new tasks, the parameters of the PANN are frozen and not trained. Only the parameters of the classifier built on the embedding features are trained. Fig. 2(b) shows this strategy, where D_{NewTask} is a new task dataset, and $\text{FC}_{\text{NewTask}}$ is the fully connected layer of a new task. The PANN is used as a feature extractor. A classifier is built on the extracted embedding features. The shaded rectangle indicates the parameters which are frozen and not trained.

(3) Fine-tune a PANN. A PANN is used for a new task, except the final fully-connected layer. All parameters are initialized from the PANN, except the final fully-connected layer which is randomly initialized. All parameters are fine-tuned on D_{NewTask}. Fig. 2(c) demonstrates the fine-tuning of a PANN.

VI. Experiments

First, we evaluate the performance of PANNs on AudioSet tagging. Then, the PANNs are transferred to several audio pattern recognition tasks, including acoustic scene classification, general audio tagging, music classification and speech emotion classification.

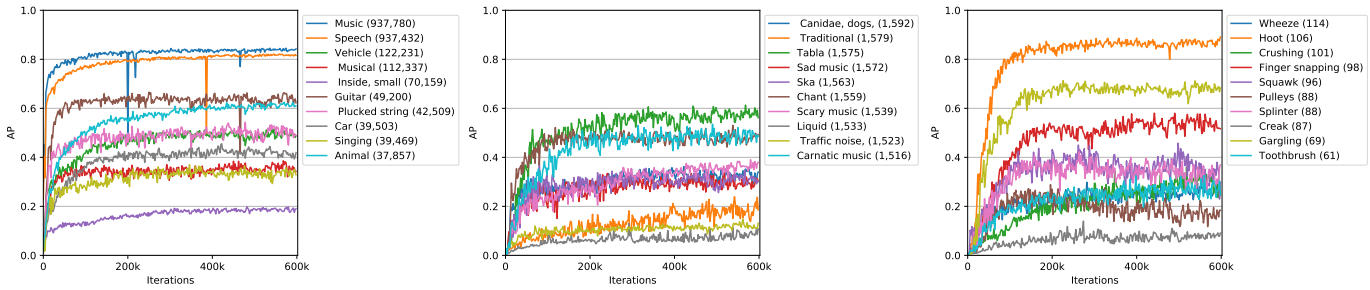


Fig. 3. Class-wise AP of sound events with the CNN14 system. The number inside parentheses indicates the number of training clips. The left, middle, right columns show the AP of sound classes with the number of training clips ranked the 1st to 10th, 250th to 260th and 517th to 527th in the training set of AudioSet.

A. AudioSet dataset

AudioSet is a large-scale audio dataset with an ontology of 527 sound classes [1]. The audio clips from AudioSet are extracted from YouTube videos. The training set consists of 2,063,839 audio clips, including a "balanced subset" of 22,160 audio clips, where there are at least 50 audio clips for each sound class. The evaluation set consists of 20,371 audio clips. Instead of using the embedding features provided by [1], we downloaded raw audio waveforms of AudioSet in Dec. 2018 using the links provided by [1], and ignored the audio clips that are no longer downloadable. We successfully download 1,934,187 (94%) of the audio clips of the full training set, including 20,550 (93%) of the audio clips of the balanced training set. We successfully download 18,887 audio clips of the evaluation dataset. We pad the audio clips to 10 seconds with silence if they are shorter than 10 seconds. Considering the fact that a large number of audio clips from YouTube are monophonic and have a low sampling rate, we convert all audio clips to monophonic and resample them to 32 kHz.

For the CNN systems based on log mel spectrograms, STFT is applied on the waveforms with a Hamming window of size 1024 [33] and a hop size of 320 samples. This configuration leads to 100 frames per second. Following [33], we apply 64 mel filter banks to calculate the log mel spectrogram. The lower and upper cut-off frequencies of the mel banks are set to 50 Hz and 14 kHz to remove low frequency noise and the aliasing effects. We use the *torchlibrosa*[1], a PyTorch implementation of functions of librosa [46] to build log mel spectrogram extraction into PANNs. The log mel spectrogram of a 10-second audio clip has a shape of 1001×64. The extra one frame is caused by applying the "centre" argument when calculating STFT. A batch size of 32, and an Adam [47] optimizer with a learning rate of 0.001 is used for training. Systems are trained on a single card Tesla-V100-PCIE-32GB. Each system takes around 3 days to train from scratch for 600 k iterations.

B. Evaluation metrics

Mean average precision (mAP), mean area under the curve (mAUC) and d-prime are used as official evaluation metrics for AudioSet tagging [20][1]. Average precision (AP) is the area under the recall and precision curve. AP does not depend on the number of true negatives, because neither precision nor recall depends on the number of true negatives. On the other hand, AUC is the area under the false positive rate and true positive rate (recall) which reflects the influence of the true negatives. The d-prime [1] is also used as an metric and be calculated directly from AUC [1]. All metrics are calculated on individual classes, and then averaged across all classes. Those metrics are also called macro metrics.

[1]https://github.com/qiuqiangkong/torchlibrosa

TABLE IV
COMPARISON WITH PREVIOUS METHODS

	mAP	AUC	d-prime
Random guess	0.005	0.500	0.000
Google CNN [1]	0.314	0.959	2.452
Single-level attention [16]	0.337	0.968	2.612
Multi-level attention [17]	0.360	0.970	2.660
Large feature-level attention [20]	0.369	0.969	2.640
TAL Net [19]	0.362	0.965	2.554
DeepRes [48]	0.392	0.971	2.682
Our proposed CNN14	**0.431**	**0.973**	**2.732**

C. AudioSet tagging results

1) Comparison with previous methods: Table IV shows the comparison of our proposed CNN14 system with previous AudioSet tagging systems. We choose CNN14 as a basic model to investigate a various of hyper-parameter configurations for AudioSet tagging, because CNN14 is a standard CNN that has a simple architecture, and can be compared with previous CNN systems [3][33]. As a baseline, random guess achieves an mAP of 0.005, an AUC of 0.500 and a d-prime of 0.000, respectively. The result released by Google [1] trained with embedding features from [13] achieved an mAP of 0.314 and an AUC of 0.959, respectively. The single-level attention and multi-level attention systems [16], [17] achieved mAPs of 0.337 and 0.360, which were later improved by a feature-level attention neural network that achieved an mAP of 0.369. Wang et al. [19] investigated five different types of attention functions and achieved an mAP of 0.362. All the above systems were built on the embedding features released with AudioSet [1]. The recent DeepRes system [48] was built on waveforms downloaded from YouTube, and achieved an mAP of 0.392. The bottom rows of Table IV shows our proposed

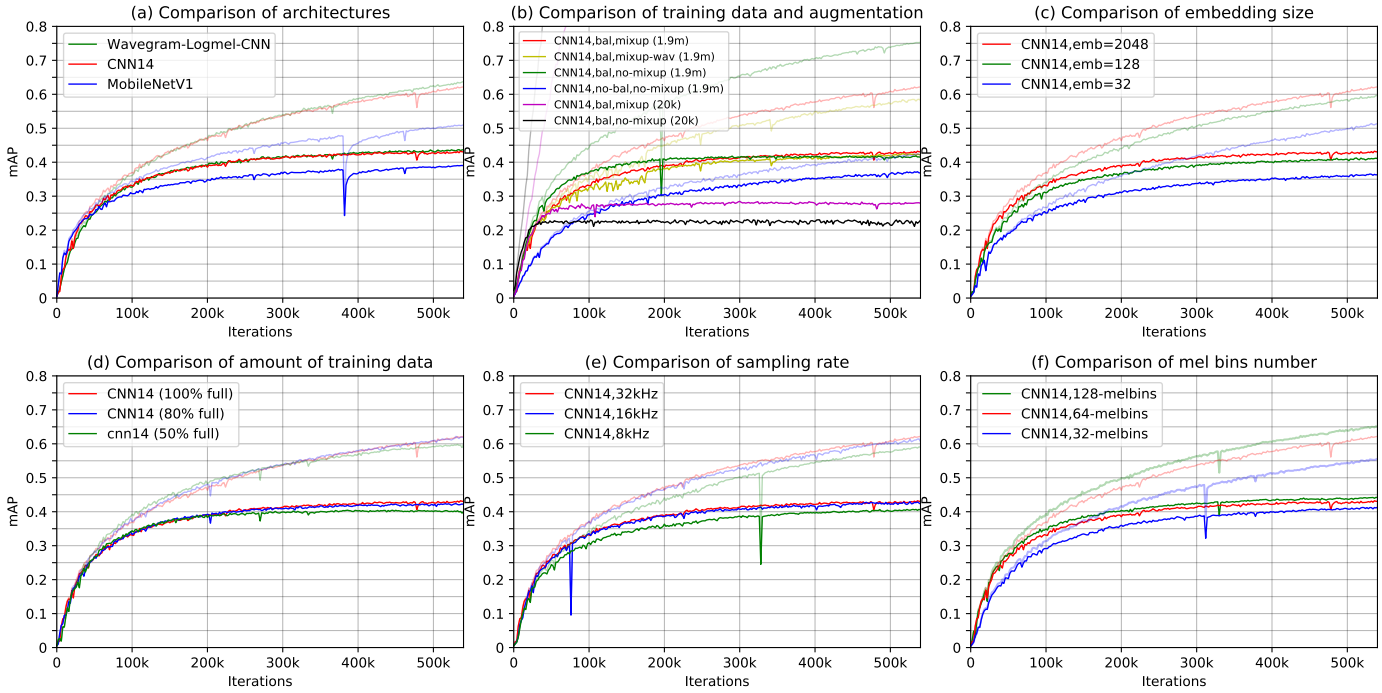


Fig. 4. Results of PANNs on AudioSet tagging. The transparent and solid lines are training mAP and evaluation mAP, respectively. The six plots show the results with different: (a) architectures; (b) data balancing and data augmentation; (c) embedding size; (d) amount of training data; (e) sampling rate; (f) number of mel bins.

CNN14 system achieves an mAP of 0.431, outperforming the best of previous systems. We use CNN14 as a backbone to build Wavegram-Logmel-CNN for fair comparison with the CNN14 system. Fig. 4(a) shows that Wavegram-Logmel-CNN outperforms the CNN14 system, and the MobileNetV1 system. Detailed results are shown in later this section in Table XI.

2) Class-wise performance: Fig. 3 shows the class-wise AP of different sound classes with the CNN14 system. The left, middle, right columns show the AP of sound classes with the number of training clips ranked the 1st to 10th, 250th to 260th and 517th to 527th in the training set of AudioSet. The performance of different sound classes can be very different. For example, "Music" and "Speech" achieve APs of over 0.80. On the other hand, some sound classes such as "Inside, small" achieve an AP of only 0.19. Fig. 3 shows that APs are usually not correlated with the number of training clips. For example, the left column shows that "Inside, small" contains 70,159 training clips, while its AP is low. In contrast, the right column shows that "Hoot" only has 106 training clips, but achieves an AP of 0.86, and is larger than many other sound classes with more training clips. In the end of this article, we plot the mAP of all 527 sound classes in Fig. 12, which shows the class-wise comparison of the CNN14, MobileNetV1 and Wavegram-Logmel-CNN systems with previous state-of-the-art audio tagging system [20] built with embedding features released by [1]. The blue bars in Fig. 12 show the number of training clips in logarithmic scale. The "+" symbol indicates label qualities between 0 and 1, which are measured by the percentage of correctly labelled audio clips verified by an expert [1]. The label quality vary from sound class to sound class. The "−" symbol indicates the sound classes whose label quality are not available. Fig. 12 shows that the average precisions of some classes are higher than others. For example, sound classes such as "bagpipes" achieve an average precision of 0.90, while sound classes such as "mouse" achieve an average precision less than 0.2. One explanation is that the audio tagging difficulty is different between sound class to sound class. In addition, audio tagging performance is not always correlated with the number of training clips and label qualities [20]. Fig. 12 shows that our proposed systems outperform previous state-of-the-art systems [16], [17] across a wide range of sound classes.

TABLE V
RESULTS WITH DATA BALANCING AND AUGMENTATION

Augmentation	mAP	AUC	d-prime
no-bal,no-mixup (20k)	0.224	0.894	1.763
bal,no-mixup (20k)	0.221	0.879	1.652
bal,mixup (20k)	**0.278**	**0.905**	**1.850**
no-bal,no-mixup (1.9m)	0.375	0.971	2.690
bal,no-mixup (1.9m)	0.416	0.968	2.613
bal,mixup (1.9m)	**0.431**	**0.973**	**2.732**
bal,mixup-wav (1.9m)	0.425	0.973	2.720

TABLE VI
RESULTS OF DIFFERENT HOP SIZES

Hop size	Time resolution	mAP	AUC	d-prime
1000	31.25 ms	0.400	0.969	2.645
640	20.00 ms	0.417	0.972	2.711
500	15.63 ms	0.417	0.971	2.682
320	10.00 ms	**0.431**	**0.973**	**2.732**

TABLE VII
RESULTS OF DIFFERENT EMBEDDING DIMENSIONS

Embedding	mAP	AUC	d-prime
32	0.364	0.958	2.437
128	0.412	0.969	2.634
512	0.420	0.971	2.689
2048	**0.431**	**0.973**	**2.732**

TABLE VIII
RESULTS OF PARTIAL TRAINING DATA

Training data	mAP	AUC	d-prime
50% of full	0.406	0.964	2.543
80% of full	0.426	0.971	2.677
100% of full	**0.431**	**0.973**	**2.732**

TABLE IX
RESULTS OF DIFFERENT SAMPLE RATES

Sample rate	mAP	AUC	d-prime
8 kHz	0.406	0.970	2.654
16 kHz	0.427	0.973	2.719
32 kHz	**0.431**	**0.973**	**2.732**

TABLE X
RESULTS OF DIFFERENT MEL BINS

Mel bins	mAP	AUC	d-prime
32 bins	0.413	0.971	2.691
64 bins	0.431	0.973	2.732
128 bins	**0.442**	**0.973**	**2.735**

3) Data balancing: Section IV-A introduces the data balancing technique that we use to train AudioSet tagging systems. Fig. 4(b) shows the performance of the CNN14 system with and without data balancing. The blue curve shows that it takes a long time to train PANNs without data balancing. The green curve shows that with data balancing, a system converges faster within limited training iterations. In addition, the systems trained with full 1.9 million training clips perform better than the systems trained with the balanced subset of 20k training clips. Table V shows that the CNN14 system achieves an mAP of 0.416 with data balancing, higher than that without data balancing (0.375).

4) Data augmentation: We show that mixup data augmentation plays an important role in training PANNs. By default, we apply mixup on the log mel spectrogram. Fig. 4(b) and Table V shows that the CNN14 system trained with mixup data augmentation achieves an mAP of 0.431, outperforming that trained without mixup data augmentation (0.416). Mixup is especially useful when training with the balanced subset containing only 20k training clips, yielding an mAP of 0.278, compared to training without mixup (0.221). In addition, we show that mixup on the log mel spectrogram achieves an mAP of 0.431, outperforming mixup in the time-domain waveform of 0.425, when training with full training data. This suggests that mixup is more effective when used with the log mel spectrogram than with the time-domain waveform.

5) Hop sizes: The hop size is the number of samples between adjacent frames. A small hop size leads to high resolution in the time domain. We investigate the influence of different hop sizes on AudioSet tagging with the CNN14 system. We investigate hop sizes of 1000, 640, 500 and 320: these correspond to time domain resolutions of 31.25 ms, 20.00 ms, 15.63 ms and 10.00 ms between adjacent frames, respectively. Table VI shows that the mAP score increases as hop sizes decrease. With a hop size of 320, the CNN14 system achieves an mAP of 0.431, outperforming the larger hop sizes of 500, 640 and 1000.

6) Embedding dimensions: Embedding features are fixed-length vectors that summarize audio clips. By default, the CNN14 has an embedding feature dimension of 2048. We investigate a rage of CNN14 systems with embedding dimensions of 32, 128, 512 and 2048. Fig. 4(c) and Table VII show that mAP performance increases as embedding dimension increases.

7) Training with partial data: The audio clips of AudioSet are sourced from YouTube. Same audio clips are no longer downloadable, and others may be removed in the future. For better reproducibility of our work in future, we investigate the performance of systems trained with randomly chosen partial data ranging from 50% to 100% of our downloaded data. Fig. 4(d) and Table VIII show that the mAP decreases slightly from 0.431 to 0.426 (a 1.2% drop) when the CNN14 system is trained with 80% of full data, and decreases to 0.406 (a 5.8% drop) when trained with 50% of full data. This result shows that the amount of training data is important for training PANNs.

8) Sample rate: Fig. 4(e) and Table IX show the performance of the CNN14 system trained with different sample rate. The CNN14 system trained with 16 kHz audio recordings achieves an mAP of 0.427, similar (within 1.0%) to the CNN14 system trained with a sample rate of 32 kHz. On the other hand, the CNN14 system trained with 8 kHz audio recordings achieves a lower mAP of 0.406 (5.8% lower). This indicates that information in the 4 kHz - 8 kHz range is useful for audio tagging.

9) Mel bins: Fig. 4(f) and Table X show the performance of the CNN14 system trained with different number of mel bins. The system achieves an mAP of 0.413 with 32 mel bins, compared to 0.431 with 64 mel bins and 0.442 with 128 mel bins. This result suggests that PANNs achieve better performance with more mel bins, although the computation complexity increases linearly with the number of mel bins. Throughout this paper, we adopt 64 mel bins for extracting the log mel spectrogram, as a trade-off between computational complexity and system performance.

10) Number of CNN layers: We investigate the performance of CNN systems with 6, 10 and 14 layers, as described in Section II-A. Table XI shows that the 6-, 10- and 14-layer CNNs achieve mAPs of 0.343, 0.380 and 0.431, respectively. This result suggests that PANNs with deeper CNN architectures achieve better performance than shallower CNN architectures. This result is in contrast to previous audio tagging systems trained on smaller datasets where shallower

TABLE XI
RESULTS OF DIFFERENT SYSTEMS

Architecture	mAP	AUC	d-prime
CNN6	0.343	0.965	2.568
CNN10	0.380	0.971	2.678
CNN14	**0.431**	**0.973**	**2.732**
ResNet22	0.430	0.973	0.270
ResNet38	**0.434**	**0.974**	**2.737**
ResNet54	0.429	0.971	2.675
MobileNetV1	**0.389**	**0.970**	**2.653**
MobileNetV2	0.383	0.968	2.624
DaiNet [31]	0.295	0.958	2.437
LeeNet11 [42]	0.266	0.953	2.371
LeeNet24	0.336	0.963	2.525
Res1dNet31	**0.365**	**0.958**	**2.444**
Res1dNet51	0.355	0.948	2.295
Wavegram-CNN	0.389	0.968	2.612
Wavegram-Logmel-CNN	**0.439**	**0.973**	**2.720**

CNNs such as 9-layer CNN performed better than deeper CNNs [33]. One possible explanation is that smaller datasets may suffer from overfitting, while AudioSet is large enough to train deeper CNNs, at least up to the 14 layers CNNs that we investigate.

11) ResNets: We apply ResNets to investigate the performance of deeper PANNs. Table XI shows that the ResNet22 system achieves an mAP of 0.430 similar to the CNN14 system. ResNet38 achieves an mAP of 0.434, slightly outperforming other systems. ResNet54 achieves an mAP of 0.429, which does not further improve the performance.

12) MobileNets: The systems mentioned above show that PANNs achieve good performance in AudioSet tagging. However, those systems do not consider computational efficiency when implemented on portable devices. We investigate building PANNs with light weight MobileNets described in Section II-C. Table XI shows that MobileNetV1 achieves an mAP of 0.389, 9.7% lower to the CNN14 system of 0.431. The number of multiplication and addition (multi-adds) and parameters of the MobileNetV1 system are only 8.6% and 5.9% of the CNN14 system, respectively. The MobileNetV2 system achieves an mAP of 0.383, 11.1% lower than CNN14, and is more computationally efficient than MobileNetV1, where the number of multi-adds and parameters are only 6.7% and 5.0% of the CNN14 system.

13) One-dimensional CNNs: Table XI shows the performance of one-dimensional CNNs. The DaiNet with 18 layers [31] achieves an mAP of 0.295. The LeeNet11 with 11 layers [42] achieves an mAP of 0.266. Our improved LeeNet with 24 layers improves the mAP of LeeNet11 to 0.336. Our proposed Res1dNet31 and Res1dNet51 described in Section II-D3 achieve mAPs of 0.365 and 0.355 respectively, and achieve state-of-the-art performance among one-dimensional CNN systems.

14) Wavegram-Logmel-CNN: The bottom rows of Table XI show the result of our proposed Wavegram-CNN and Wavegram-Logmel-CNN systems. The Wavegram-CNN system achieves an mAP of 0.389, outperforming the best previous one-dimensional CNN system (Res1dNet31). This indicates that the Wavegram is an effective learnt feature. Furthermore, our proposed Wavegram-Logmel-CNN system achieves a state-of-the-art mAP of 0.439 among all PANNs.

TABLE XII
NUMBER OF MULTI-ADDS AND PARAMETERS OF DIFFERENT SYSTEMS

Architecture	Multi-Adds	Parameters
CNN6	21.986 $\times 10^9$	4,837,455
CNN10	28.166 $\times 10^9$	5,219,279
CNN14	42.220 $\times 10^9$	80,753,615
ResNet22	30.081 $\times 10^9$	63,675,087
ResNet38	48.962 $\times 10^9$	73,783,247
ResNet54	54.563 $\times 10^9$	104,318,159
MobileNetV1	3.614 $\times 10^9$	4,796,303
MobileNetV2	2.810 $\times 10^9$	4,075,343
DaiNet	30.395 $\times 10^9$	4,385,807
LeeNet11	4.741 $\times 10^9$	748,367
LeeNet24	26.369 $\times 10^9$	10,003,791
Res1dNet31	32.688 $\times 10^9$	80,464,463
Res1dNet51	61.833 $\times 10^9$	106,538,063
Wavegram-CNN	44.234 $\times 10^9$	80,991,759
Wavegram-Logmel-CNN	53.510 $\times 10^9$	81,065,487

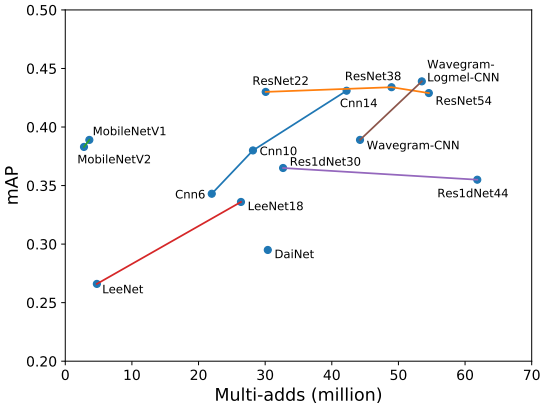


Fig. 5. Multi-adds versus mAP of AudioSet tagging systems. The same types of architectures are grouped in the same color.

15) Complexity analysis: We analyze the computational complexity of PANNs for inference. The number of multi-adds and parameters are two important factors for complexity analysis. The middle column of Table XII shows the number of multi-adds to infer a 10-second audio clip. The right column of Table XII shows the number of parameters of different systems. The number of multi-adds and parameters of the CNN14 system are 42.2×10^9 and 80.8 million, respectively, which are larger than the CNN6 and CNN10 systems. The number of multi-adds for the ResNets22 and ResNet38 systems are slightly less than for the CNN14 system. The ResNet54 system contains the most multi-adds at 54.6×10^9. One-dimensional CNNs have a similar computational cost to the two-dimensional CNNs. The best performing one-dimensional system, Res1dNet31, contains 32.7×10^9 multi-adds and 80.5 million parameters. Our proposed Wavegram-CNN system contains 44.2×10^9 multi-adds and 81.0 million parameters, which is similar to CNN14. The Wavegram-Logmel-CNN system slightly increases the multi-adds to 53.5×10^9, and the number of parameters is to 81.1 million, which is similar to CNN14. To reduce the number of multi-adds and parameters,

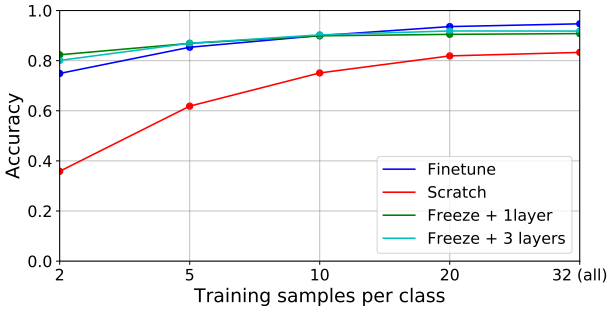


Fig. 6. Accuracy of ESC-50 with various number of training clips per class.

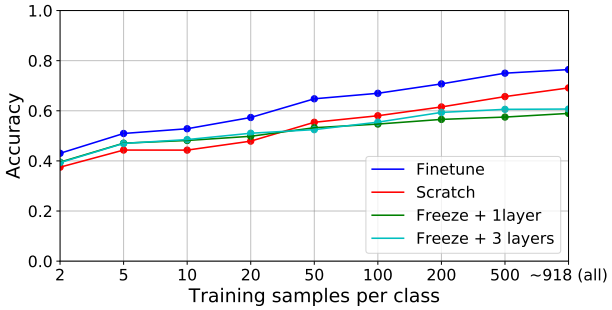


Fig. 7. Accuracy of DCASE 2019 Task 1 with various number of training clips per class.

TABLE XIII
ACCURACY OF ESC-50

	STOA [49]	Scratch	fine-tune	Freeze_L1	Freeze_L3
Acc.	0.865	0.833	**0.947**	0.908	0.918

TABLE XIV
ACCURACY OF DCASE 2019 TASK 1

	STOA [51]	Scratch	Fine-tune	Freeze_L1	Freeze_L3
Acc.	**0.851**	0.691	0.764	0.589	0.607

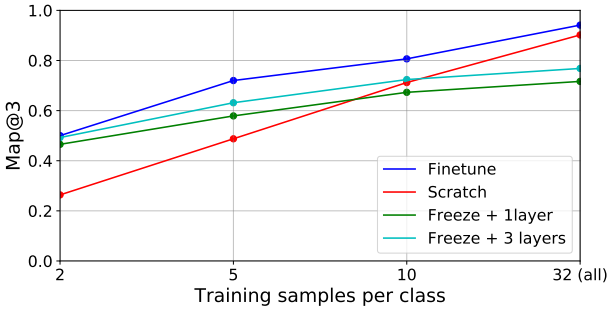


Fig. 8. Accuracy of DCASE 2018 Task 2 with various number of training clips per class.

TABLE XV
ACCURACY OF DCASE 2018 TASK 2

	STOA [52]	Scratch	Fine-tune	Freeze_L1	Freeze_L3
mAP@3	**0.954**	0.902	0.941	0.717	0.768

MobileNets are applied. The MobileNetV1 and MobileNetV2 systems are light weight CNNs, with only 3.6×10^9 and 2.8×10^9 multi-adds and around 4.8 million and 4.1 million parameters, respectively. MobileNets reduce both the computational cost and system size. Figure 5 summarizes the mAP versus multi-adds of different PANNs. The same type of systems are linked by lines of the same color. The mAP increases from bottom to top. On the top-right is our proposed Wavegram-Logmel-CNN system that achieves the best mAP. On the top-left are MobileNetV1 and MobileNetV2 that are the most computationally efficient systems.

D. Transfer to other tasks

In this section, we investigate the application of PANNs to a range of other pattern recognition tasks. PANNs can useful for few-shot learning, for the tasks where only a limited number of training clips are provided. Few-shot learning is an important research topic in audio pattern recognition, as collecting labelled data can be time consuming. We transfer PANNs to other audio pattern recognition tasks using the methods described in Section V. To begin with, we resample all audio recordings to 32 kHz, and convert them to monophonic to be consistent with the PANNs trained on AudioSet. We perform the following strategies described in Section V for each task: 1) Train a system from scratch; 2) Use a PANN as a feature extractor; 3) Fine-tune a PANN. When using a PANN as the feature extractor, we build classifiers on the embedding features with one and three fully-connected layers, and call them "Freeze + 1 layers" (Freeze_L1) and "Freeze + 3 layers" (Freeze_L3), respectively. We adopt the CNN14 system for transfer learning to provide a fair comparison with other CNN based systems for audio pattern recognition. We also investigate the performance of PANNs trained with different number of shots when training other audio pattern recognition tasks.

1) ESC-50: ESC-50 is an environmental sound dataset [50] consisting of 50 sound events, such as "Dog" and "Rain". There are 2,000 5-second audio clips in the dataset, with 40 clips per class. Table XIII shows the 5-fold cross validation [50] accuracy of the CNN14 system. Sailor et al. [49] proposed a state-of-the-art system for ESC-50, achieved an accuracy of 0.865 using unsupervised filterbank learning with a convolutional restricted Boltzmann machine. Our fine-tuned system achieves an accuracy of 0.947, outperforming previous state-of-the-art system by a large margin. The Freeze_L1 and Freeze_L3 systems achieve accuracies of 0.918 and 0.908, respectively. Training the CNN14 system from scratch achieves an accuracy of 0.833. Fig. 6 shows the accuracy of ESC-50 with different numbers of training clips of each sound class. Using a PANN as a feature extractor achieves the best performance when fewer than 10 clips per sound class are available for training. With more training clips, the fine-tuned systems achieve better performance. Both the fine-tuned system and the system using the PANN as a feature extractor outperform the systems trained from scratch.

2) DCASE 2019 Task 1: DCASE 2019 Task 1 is an acoustic scene classification task [2], with a dataset consisting of over

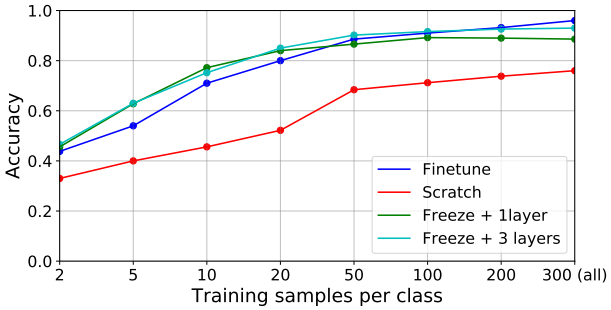


Fig. 9. Accuracy of MSoS with various number of training clips per class.

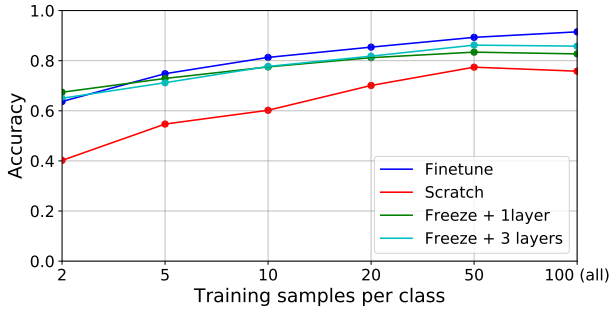


Fig. 10. Accuracy of GTZAN with various number of training clips per class.

TABLE XVI
ACCURACY OF MSOS

	STOA [53]	Scratch	Fine-tune	Freeze_L1	Freeze_L3
Acc.	0.930	0.760	**0.960**	0.886	0.930

TABLE XVII
ACCURACY OF GTZAN

	STOA [56]	Scratch	Fine-tune	Freeze_L1	Freeze_L3
Acc.	**0.939**	0.758	0.915	0.827	0.858

40 hours of stereo recordings collected from various acoustic scenes in 12 European cities. We focus on Subtask A, where each audio recording has two channels with a sampling rate of 48 kHz. In the development set, there are 9185 and 4185 audio clips for training and validation respectively. We convert the stereo recordings to monophonic by averaging the stereo channels. CNN14 trained from scratch achieves an accuracy of 0.691, while the fine-tuned system achieves an accuracy of 0.764. Freeze_L1 and Freeze_L3 achieve accuracies of 0.689 and 0.607 respectively, and do not outperform the CNN14 trained from scratch. One possible explanation for this underperformance is that the audio recordings of acoustic scene classification have different distribution of AudioSet. So using PANN as a feature extractor does not outperform fine-tune or train a system from scratch. The fine-tuned system achieves better performance than the system trained from scratch. Fig. 7 shows the classification accuracy of systems with various numbers of training clips per class. Table XIV shows the overall performance. The state-of-the-art system of Chen et al [51]. achieves an accuracy of 0.851 using the combination of various classifiers and stereo recordings as input, while we do not use any ensemble methods and stereo recordings.

3) DCASE 2018 Task 2: DCASE 2018 Task 2 is a general-purpose automatic audio tagging task [54] using a dataset of audio recordings from Freesound annotated with a vocabulary of 41 labels from the AudioSet ontology. The development set consists of 9,473 audio recordings with durations from 300 ms to 30 s. The mAP@3 is used for evaluating system performance [54]. In training, we break or pad audio recordings into 4-second audio segments. In inference, we average the predictions of those segments to obtain the prediction of an audio recording. Table XV shows that the best previous method proposed by Jeong and Lim [52] achieves an mAP@3 of 0.954 using an ensemble of several systems. In comparison, our CNN14 system trained from scratch achieves an accuracy of 0.902. The fine-tuned CNN14 system achieves an mAP@3 of 0.941. The Freeze_L1 and Freeze_L3 systems achieve accuracies of 0.717 and 0.768 respectively. Fig. 8 shows the mAP@3 with different numbers of training clips. The fine-tuned CNN14 system outperforms the systems trained from scratch and the systems using PANN as a feature extractor. The fine-tuned CNN14 system achieves comparable results to the state-of-the-art system.

4) MSoS: The Making Sense of Sounds (MSoS) data challenge [55] is a task to predict an audio recording to one of five categories: “Nature”, “Music”, “Human”, “Effects” and “Urban”. The dataset consists of a development set of 1,500 audio clips and an evaluation set of 500 audio clips. All audio clips have a duration of 4 seconds. The state-of-the-art system proposed by Chen and Gupta [53] achieves an accuracy of 0.930. Our fine-tuned CNN14 achieves an accuracy of 0.960, outperforming previous state-of-the-art system. CNN14 trained from scratch achieves an accuracy of 0.760. Fig. 9 shows the accuracy of the systems with different number of training clips. The fine-tuned CNN14 system and the system using CNN14 as a feature extractor outperforms CNN14 trained from scratch.

5) GTZAN: The GTZAN dataset [57] is a music genre classification dataset containing 1,000 30-second music clips of 10 genres of music, such as “Classical” and “Country”. All music clips have a duration of 30 seconds and a sampling rate of 22,050 Hz. In development, 10-fold cross validation is used to evaluate the performance of systems. Table XVII shows that previous state-of-the-art system proposed by Liu et al. [56] achieves an accuracy of 0.939 using a bottom-up broadcast neural network. The fine-tuned CNN14 system achieves an accuracy of 0.915, outperforming CNN14 trained from scratch with an accuracy of 0.758 and the Freeze_L1 and Freeze_L3 systems with accuracies of 0.827 and 0.858 respectively. Fig. 10 shows the accuracy of systems with different numbers of training clips. The Freeze_L1 and Freeze_L3 systems achieve better performance than other systems trained with less than 10 clips per class. With more training clips, the fine-tuned CNN14 system performs better than other systems.

6) RAVDESS: The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) is a human speech emotion dataset [59]. The database consists of sounds from

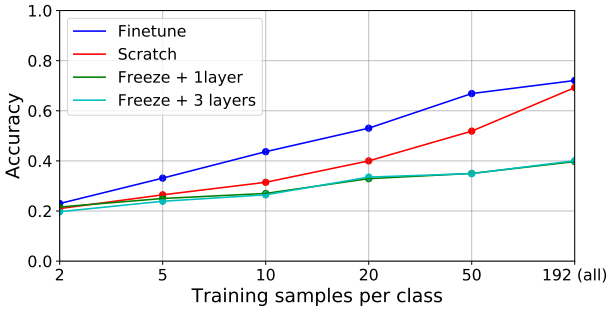


Fig. 11. Accuracy of RAVDESS with various number of training clips per class.

TABLE XVIII
ACCURACY OF RAVDESS

	STOA [58]	Scratch	Fine-tune	Freeze_L1	Freeze_L3
Acc.	0.645	0.692	**0.721**	0.397	0.401

24 professional actors including 12 female and 12 male simulating 8 emotions, such as "Happy" and "Sad". The task is to classify each sound clip into an emotion. There are 1,440 audio clips in the development set. We evaluate our systems with 4-fold cross validation. Table XVIII shows that previous state-of-the-art system proposed by Zeng et al. [58] achieves an accuracy of 0.645. Our CNN14 system trained from scratch achieves an accuracy of 0.692. The fine-tuned CNN14 system achieves a state-of-the-art accuracy of 0.721. The Freeze_L1 and Freeze_L3 systems achieve much lower accuracies of 0.397 and 0.401 respectively. Fig. 11 shows the accuracy of systems with respect to a range of training clips. The fine-tuned systems and the system trained from scratch outperform the system using PANN as a feature extractor. This suggests that audio recordings of the RAVDESS dataset may have different distributions of the AudioSet dataset. Therefore, the parameters of a PANN need be fine-tuned to achieve good performance on the RAVDESS classification task.

E. Discussion

In this article, we have investigated a wide range of PANNs for AudioSet tagging. Several of our proposed PANNs have outperformed previous state-of-the-art AudioSet tagging systems, including CNN14 achieves an mAP of 0.431, and ResNet38 achieves an mAP of 0.434, outperforming Google's baseline of 0.314. MobileNets are light-weight systems that have fewer multi-adds and numbers of parameters. MobileNetV1 achieves an mAP of 0.389. Our adapted one-dimensional system Res1dNet31 achieves an mAP of 0.365, outperforming previous one-dimensional CNNs including DaiNet [31] of 0.295 and LeeNet11 [42] of 0.266. Our proposed Wavegram-Logmel-CNN system achieves the highest mAP of 0.439 among all PANNs. PANNs can be used as a pretrained model for new audio pattern recognition tasks.

PANNs trained on the AudioSet dataset were transferred to six audio pattern recognition tasks. We show that fine-tuned PANNs achieve state-of-the-art performance in the ESC-50, MSOS and RAVDESS classification tasks, and approach the state-of-the-art performance in the DCASE 2018 Task 2 and the GTZAN classification task. Of the PANN systems, the fine-tuned PANNs always outperform PANNs trained from scratch on new tasks. The experiments show that PANNs have been successful in generalizing to other audio pattern recognition tasks with limited number of training data.

VII. CONCLUSION

We have presented pretrained audio neural networks (PANNs) trained on the AudioSet for audio pattern recognition. A wide range of neural networks are investigated to build PANNs. We propose a Wavegram feature learnt from waveform, and a Wavegram-Logmel-CNN that achieves state-of-the-art performance in AudioSet tagging, archiving an mAP of 0.439. We also investigate the computational complexity of PANNs. We show that PANNs can be transferred to a wide range of audio pattern recognition tasks and outperform several previous state-of-the-art systems. PANNs can be useful when fine-tuned on a small amount of data on new tasks. In the future, we will extend PANNs to more audio pattern recognition tasks.

REFERENCES

[1] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio Set: An ontology and human-labeled dataset for audio events," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 776–780.

[2] A. Mesaros, T. Heittola, and T. Virtanen, "A multi-device dataset for urban acoustic scene classification," in *Workshop on Detection and Classification of Acoustic Scenes and Events (DCASE)*, 2018, pp. 9–13.

[3] K. Choi, G. Fazekas, and M. Sandler, "Automatic tagging using deep convolutional neural networks," in *Conference of the International Society for Music Information Retrieval (ISMIR)*, 2016, pp. 805–811.

[4] E. Cakir, T. Heittola, H. Huttunen, and T. Virtanen, "Polyphonic sound event detection using multi label deep neural networks," in *International Joint Conference on Neural Networks (IJCNN)*, 2015.

[5] J. P. Woodard, "Modeling and classification of natural sounds by product code hidden Markov models," *IEEE Transactions on Signal Processing*, vol. 40, pp. 1833–1835, 1992.

[6] D. P. W. Ellis, "Detecting alarm sounds," https://academiccommons.columbia.edu/doi/10.7916/D8F19821/download, 2001.

[7] D. Stowell, D. Giannoulis, E. Benetos, M. Lagrange, and M. D. Plumbley, "Detection and classification of acoustic scenes and events," *IEEE Transactions on Multimedia*, vol. 17, pp. 1733–1746, 2015.

[8] A. Mesaros, T. Heittola, E. Benetos, P. Foster, M. Lagrange, T. Virtanen, and M. D. Plumbley, "Detection and classification of acoustic scenes and events: Outcome of the DCASE 2016 challenge," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 26, pp. 379–393, 2018.

[9] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, "DCASE 2017 challenge setup: Tasks, datasets and baseline system," in *Workshop on Detection and Classification of Acoustic Scenes and Events (DCASE)*, 2017, pp. 85–92.

[10] "DCASE Challenge 2019," http://dcase.community/challenge2019, 2019.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018, pp. 4171–4186.

[13] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold *et al.*, "CNN architectures for large-scale audio classification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 131–135.

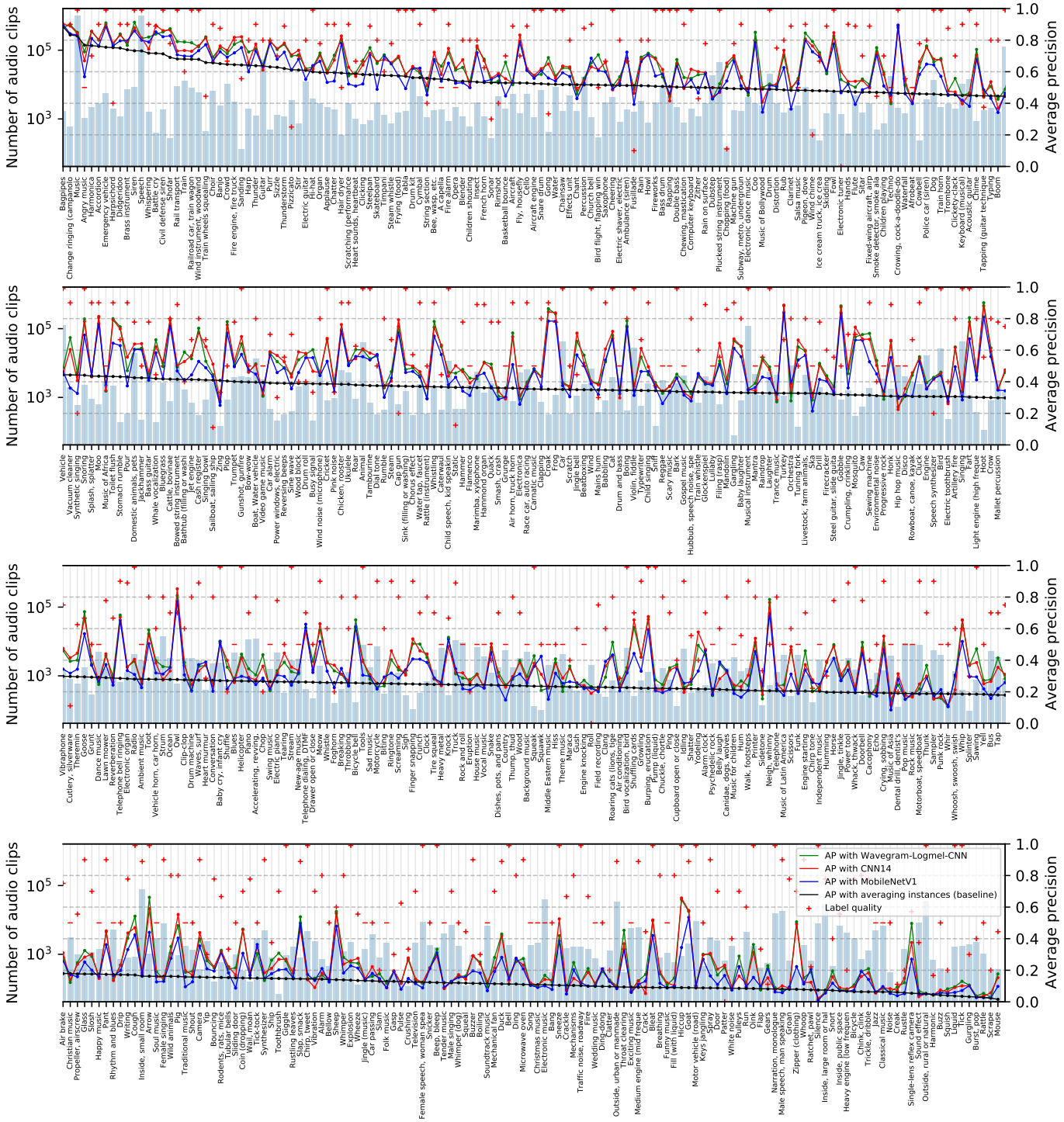


Fig. 12. Class-wise performance of AudioSet tagging systems. Red, blue and black curves are APs of CNN14, MobileNetV1 and the audio tagging system [20]. The blue bars show the number of training clips in logarithmic scale.

[14] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Transfer learning for music classification and regression tasks," in *Conference of the International Society of Music Information Retrieval (ISMIR)*, 2017, pp. 141–149.

[15] J. Pons, O. Nieto, M. Prockup, E. Schmidt, A. Ehmann, and X. Serra, "End-to-end learning for music audio tagging at scale," in *Conference of the International Society for Music Information Retrieval (ISMIR)*, 2017, pp. 637–644.

[16] Q. Kong, Y. Xu, W. Wang, and M. D. Plumbley, "Audio Set classification with attention model: A probabilistic perspective," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 316–320.

[17] C. Yu, K. S. Barsim, Q. Kong, and B. Yang, "Multi-level attention model for weakly supervised audio classification," in *Detection and Classification of Acoustic Scenes and Events (DCASE)*, 2018, pp. 188–192.

[18] S.-Y. Chou, J.-S. R. Jang, and Y.-H. Yang, "Learning to recognize transient sound events using attentional supervision," in *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018, pp. 3336–3342.

[19] Y. Wang, J. Li, and F. Metze, "A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling,"

in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 31–35.

[20] Q. Kong, C. Yu, Y. Xu, T. Iqbal, W. Wang, and M. D. Plumbley, "Weakly labelled audioset tagging with attention neural networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, pp. 1791–1802, 2019.

[21] A. Van Den Oord, S. Dieleman, and B. Schrauwen, "Transfer learning by supervised pre-training for audio-based music classification," in *Conference of the International Society for Music Information Retrieval (ISMIR)*, 2014, pp. 29–34.

[22] Y. Wang, "Polyphonic sound event detection with weak labeling," *PhD thesis, Carnegie Mellon University*, 2018.

[23] E. Law and L. Von Ahn, "Input-agreement: a new mechanism for collecting data using human computation games," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1197–1206.

[24] A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection," in *Conference on European Signal Processing Conference (EUSIPCO)*, 2016, pp. 1128–1132.

[25] J. Pons and X. Serra, "MUSICNN: Pre-trained convolutional neural networks for music audio tagging," *arXiv preprint arXiv:1909.06654*, 2019.

[26] A. Diment and T. Virtanen, "Transfer learning of weakly labelled audio," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2017, pp. 6–10.

[27] D. Li, I. K. Sethi, N. Dimitrova, and T. McGee, "Classification of general audio data for content-based retrieval," *Pattern Recognition Letters*, vol. 22, pp. 533–544, 2001.

[28] L. Vuegen, B. Broeck, P. Karsmakers, J. F. Gemmeke, B. Vanrumste, and H. Hamme, "An MFCC-GMM approach for event detection and classification," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2013.

[29] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen, "Acoustic event detection in real life recordings," in *European Signal Processing Conference (EUSIPCO)*, 2010, pp. 1267–1271.

[30] B. Uzkent, B. D. Barkana, and H. Cevikalp, "Non-speech environmental sound classification using SVMs with a new set of features," *International Journal of Innovative Computing, Information and Control*, vol. 8, pp. 3511–3524, 2012.

[31] W. Dai, C. Dai, S. Qu, J. Li, and S. Das, "Very deep convolutional neural networks for raw waveforms," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 421–425.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[33] Q. Kong, Y. Cao, T. Iqbal, Y. Xu, W. Wang, and M. D. Plumbley, "Cross-task learning for audio tagging, sound event detection and spatial localization: DCASE 2019 baseline systems," *arXiv preprint arXiv:1904.03476*, 2019.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2012, pp. 1097–1105.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning (ICML)*, 2010, pp. 807–814.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[39] M. Lin, Q. Chen, and S. Yan, "Network in network," in *International Conference on Learning Representations (ICLR)*, 2014.

[40] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[41] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[42] J. Lee, J. Park, K. L. Kim, and J. Nam, "Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms," in *Sound and Music Computing Conference*, 2017, pp. 220–226.

[43] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *Proceedings of the ACM International Conference on Multimedia*, 2014, pp. 1041–1044.

[44] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations (ICLR)*, 2018.

[45] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A simple data augmentation method for automatic speech recognition," in *INTERSPEECH*, 2019.

[46] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the Python in Science Conference*, vol. 8, 2015, pp. 18–25.

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[48] L. Ford, H. Tang, F. Grondin, and J. Glass, "A deep residual network for large-scale acoustic scene analysis," *INTERSPEECH*, pp. 2568–2572, 2019.

[49] H. B. Sailor, D. M. Agrawal, and H. A. Patil, "Unsupervised filterbank learning using convolutional restricted Boltzmann machine for environmental sound classification," in *INTERSPEECH*, 2017, pp. 3107–3111.

[50] K. J. Piczak, "ESC: Dataset for environmental sound classification," in *ACM International Conference on Multimedia*, 2015, pp. 1015–1018.

[51] H. Chen, Z. Liu, Z. Liu, P. Zhang, and Y. Yan, "Integrating the data augmentation scheme with various classifiers for acoustic scene modeling," *DCASE2019 Challenge, Tech. Rep.*, 2019.

[52] I.-Y. Jeong and H. Lim, "Audio tagging system for DCASE 2018: focusing on label noise data augmentation and its efficient learning," *DCASE Challenge Tech. Rep.*, 2018.

[53] T. Chen and U. Gupta, "Attention-based convolutional neural network for audio event classification with feature transfer learning," *https://cvssp.org/projects/making_sense_of_sounds/site/assets/challenge_abstracts_and_figures/Tianxiang_Chen.pdf*, 2018.

[54] E. Fonseca, M. Plakal, F. Font, D. P. W. Ellis, X. Favory, J. Pons, and X. Serra, "General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline," in *Workshop on Detection and Classification of Acoustic Scenes and Events (DCASE)*, November 2018, pp. 69–73.

[55] C. Kroos, O. Bones, Y. Cao, L. Harris, P. J. Jackson, W. J. Davies, W. Wang, T. J. Cox, and M. D. Plumbley, "Generalisation in environmental sound classification: The 'Making Sense of Sounds' data set and challenge," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 8082–8086.

[56] C. Liu, L. Feng, G. Liu, H. Wang, and S. Liu, "Bottom-up broadcast neural network for music genre classification," *arXiv preprint arXiv:1901.08928*, 2019.

[57] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, pp. 293–302, 2002.

[58] Y. Zeng, H. Mao, D. Peng, and Z. Yi, "Spectrogram based multi-task audio classification," *Multimedia Tools and Applications*, vol. 78, pp. 3705–3722, 2019.

[59] S. R. Livingstone, K. Peck, and F. A. Russo, "RAVDESS: The Ryerson audio-visual database of emotional speech and song," in *Annual Meeting of the Canadian Society for Brain, Behaviour and Cognitive Science (CSBBCS)*, 2012, pp. 1459–1462.

Qiuqiang Kong (S'17) received the B.Sc. degree in 2012, and the M.E. degree in 2015 from South China University of Technology, Guangzhou, China. He received the Ph.D. degree from University of Surrey, Guildford, UK in 2020. Following his PhD, he joined ByteDance AI Lab as a research scientist. His research topic includes the classification, detection and separation of general sounds and music. He is known for developing attention neural networks for audio tagging, and winning the audio tagging task in the detection and classification of acoustic scenes and events (DCASE) challenge in 2017. He was nominated as the postgraduate research student of the year in University of Surrey, 2019. He is a frequent reviewer for journals and conferences in the area including IEEE/ACM Transactions on Audio, Speech, and Language Processing.

Yin Cao (M'18) received the B.Sc. degree in Electronic Science and Engineering from Nanjing University, China in 2008, and Ph.D. degree from Institute of Acoustics, Chinese Academy of Sciences, China in 2013. He then worked in Acoustics group at Brigham Young University, US, and at Institute of Acoustics, Chinese Academy of Sciences, China. In 2018, he joined Centre for Vision, Speech and Signal Processing at the University of Surrey. His research topic includes active noise control, air acoustics and signal processing, detection, classification and separation of audio. He is known for his work on decentralized active noise control, weighted spatial gradients control metric, and polyphonic sound event detection and localization. He was the winner of urban sound tagging in detection and classification of acoustic scenes and events (DCASE) 2020 challenge and achieved second-best of sound event detection and localization tasks in DCASE 2019 challenge. He has served as an Associate Editor for Noise Control Engineering Journal since 2020. He is also a frequent reviewer for IEEE/ACM Transactions on Audio, Speech, and Language Processing.

Turab Iqbal received the B.Eng. degree in Electronic Engineering from the University of Surrey, U.K., in 2017. Currently, he is working towards a Ph.D. degree from the Centre for Vision, Speech and Signal Processing (CVSSP) in the University of Surrey. During his time as a Ph.D. student, he has worked on a number of projects in the area of audio classification and localization using machine learning methods. His research is mainly focused on learning with weakly-labeled or noisy training data.

Wenwu Wang (M'02-SM'11) was born in Anhui, China. He received the B.Sc. degree in 1997, the M.E. degree in 2000, and the Ph.D. degree in 2002, all from Harbin Engineering University, China. He then worked in King's College London, Cardiff University, Tao Group Ltd. (now Antix Labs Ltd.), and Creative Labs, before joining University of Surrey, UK, in May 2007, where he is currently a professor in signal processing and machine learning, and a Co-Director of the Machine Audition Lab within the Centre for Vision Speech and Signal Processing. He has been a Guest Professor at Qingdao University of Science and Technology, China, since 2018. His current research interests include blind signal processing, sparse signal processing, audio-visual signal processing, machine learning and perception, machine audition (listening), and statistical anomaly detection. He has (co)-authored over 200 publications in these areas. He served as an Associate Editor for IEEE Transactions on Signal Processing from 2014 to 2018. He is also Publication Co-Chair for ICASSP 2019, Brighton, UK. He currently serves as Senior Area Editor for IEEE Transactions on Signal Processing and an Associate Editor for IEEE/ACM Transactions on Audio Speech and Language Processing.

Mark D. Plumbley (S'88-M'90-SM'12-F'15) received the B.A.(Hons.) degree in electrical sciences and the Ph.D. degree in neural networks from University of Cambridge, Cambridge, U.K., in 1984 and 1991, respectively. Following his PhD, he became a Lecturer at King's College London, before moving to Queen Mary University of London in 2002. He subsequently became Professor and Director of the Centre for Digital Music, before joining the University of Surrey in 2015 as Professor of Signal Processing. He is known for his work on analysis and processing of audio and music, using a wide range of signal processing techniques, including matrix factorization, sparse representations, and deep learning. He is a co-editor of the recent book on Computational Analysis of Sound Scenes and Events, and Co-Chair of the recent DCASE 2018 Workshop on Detection and Classifications of Acoustic Scenes and Events. He is a Member of the IEEE Signal Processing Society Technical Committee on Signal Processing Theory and Methods, and a Fellow of the IET and IEEE.