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

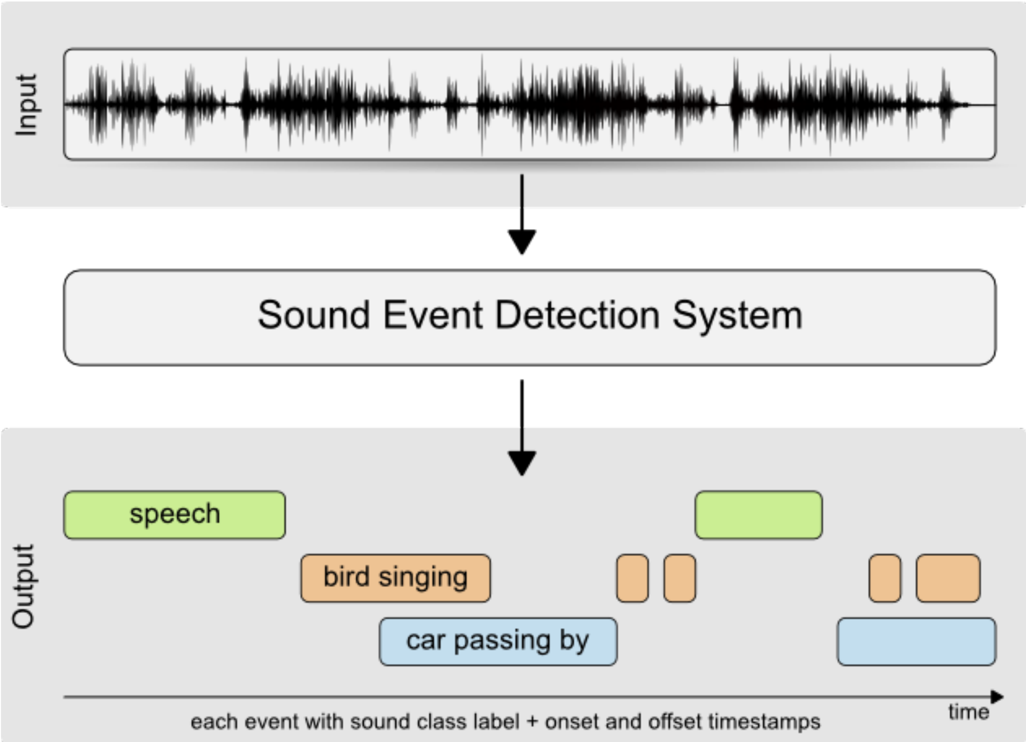
import packages

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)](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)](https://www.kaggle.com/hidehisaarai1213/introduction-to-sound-event-detection)

PANN Utils

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

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

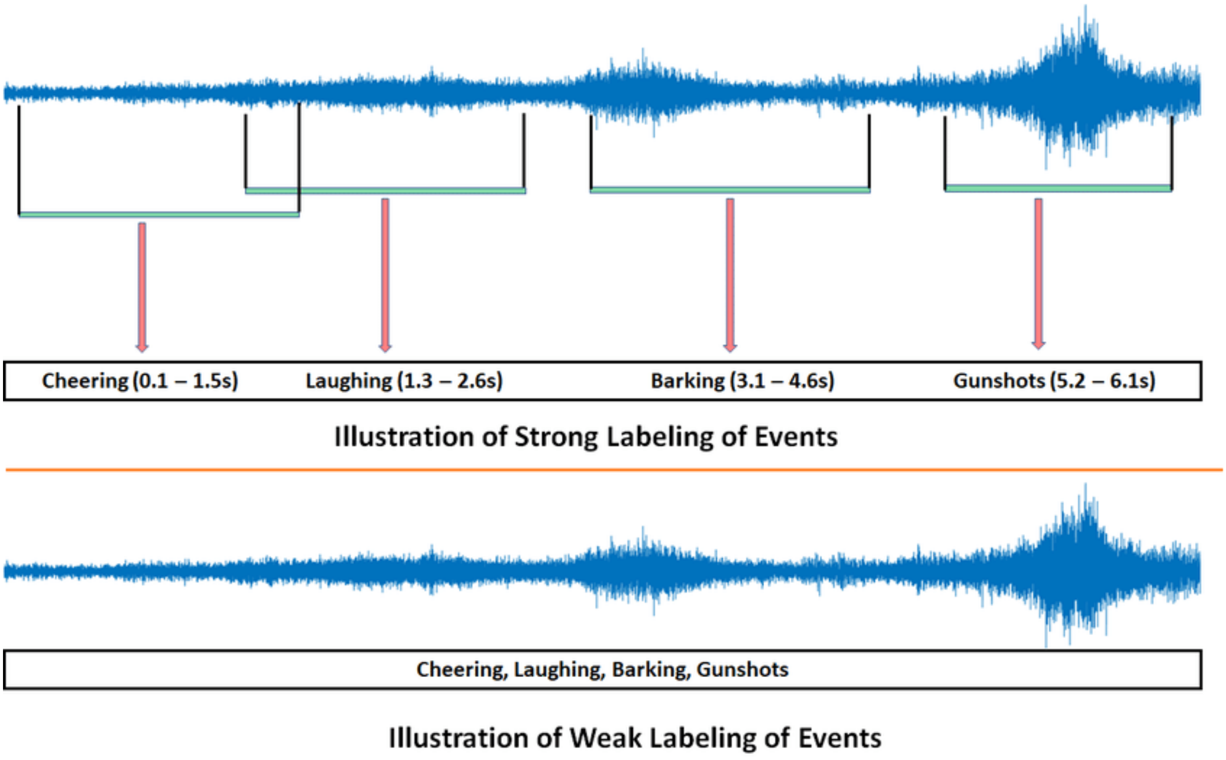


Illustration of Strong Labeling of Events

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]
```