Import additional packages

In [ ]:

```
# https://github.com/rwightman/gen-efficientnet-pytorch
!pip install ../input/geffnet-pack/gen-efficientnet-pytorch-master/ > /dev/null
# https://github.com/zhanghang1989/ResNeSt
!pip install ../input/resnest-git/ResNeSt-master/ > /dev/null
```

Import packages

In [ ]:

```
import math

from glob import glob
from os.path import join as pjoin
from tqdm.notebook import tqdm
from typing import Mapping, Any, List, Tuple, Optional, Union
from pathlib import Path

import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import pandas as pd
import librosa
import geffnet

from resnest.torch import resnest50_fast_1s1x64d
from torchvision.models.resnet import ResNet, Bottleneck
```

Main Config Variables

In [ ]:

```
# Sample rate of audio
TARGET_SR=32_000
# Whether to normalize audio by max(np.abs())
PP_NORMALIZE=True
# Minimum length of audio to predict
MIN_SEC = 1
# Computational device
DEVICE = 'cuda'
# Inference batch size
BATCH_SIZE = 64
# Thresholding value
SIGMOID_THRESH = 0.3
# Maximum amount of birds in raw. If None - no limit
MAX_BIRDS = None
```

In [ ]:

```
# Bird code dict

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': 104,
    'gockin': 105, 'gocspa': 106, 'goleag': 107, 'grbher3': 108, 'grcfly': 109,
    'greegr': 110, 'greroa': 111, 'greyel': 112, 'grhowl': 113, 'grnher': 114,
    'grtgra': 115, 'grycat': 116, 'gryfly': 117, 'haiwoo': 118, 'hamfly': 119,
    'hergul': 120, 'herthr': 121, 'hoomer': 122, 'hoowar': 123, 'horgre': 124,
    'horlar': 125, 'houfin': 126, 'houspa': 127, 'houwre': 128, 'indbun': 129,
    'juntit1': 130, 'killde': 131, 'labwoo': 132, 'larspa': 133, 'lazbun': 134,
    'leabit': 135, 'leafly': 136, 'leasan': 137, 'lecthr': 138, 'lesgol': 139,
    'lesnig': 140, 'lesyel': 141, 'lewwoo': 142, 'linspa': 143, 'lobcur': 144,
    'lobdow': 145, 'logshr': 146, 'lotduc': 147, 'louwat': 148, 'macwar': 149,
    'magwar': 150, 'mallar3': 151, 'marwre': 152, 'merlin': 153, 'moublu': 154,
    'mouchi': 155, 'moudov': 156, 'norcar': 157, 'norfli': 158, 'norhar2': 159,
    'normoc': 160, 'norpar': 161, 'norpin': 162, 'norsho': 163, 'norwat': 164,
    'nrwswa': 165, 'nutwoo': 166, 'olsfly': 167, 'orcwar': 168, 'osprey': 169,
    'ovenbi1': 170, 'palwar': 171, 'pasfly': 172, 'pecsan': 173, 'perfal': 174,
    'phaino': 175, 'pibgre': 176, 'pilwoo': 177, 'pingro': 178, 'pinjay': 179,
    'pinsis': 180, 'pinwar': 181, 'plsvir': 182, 'prawar': 183, 'purfin': 184,
    'pygnut': 185, 'rebmer': 186, 'rebnut': 187, 'rebsap': 188, 'rebwoo': 189,
    'redcro': 190, 'redhea': 191, 'reevir1': 192, 'renpha': 193, 'reshaw': 194,
    'rethaw': 195, 'rewbla': 196, 'ribgul': 197, 'rinduc': 198, 'robgro': 199,
    'rocpig': 200, 'rocwre': 201, 'rthhum': 202, 'ruckin': 203, 'rudduc': 204,
    'rufgro': 205, 'rufhum': 206, 'rusbla': 207, 'sagspa1': 208, 'sagthr': 209,
    'savspa': 210, 'saypho': 211, 'scatan': 212, 'scoori': 213, 'semplo': 214,
    'semsan': 215, 'sheowl': 216, 'shshaw': 217, 'snobun': 218, 'snogoo': 219,
    'solsan': 220, 'sonspa': 221, 'sora': 222, 'sposan': 223, 'spotow': 224,
    'stejay': 225, 'swahaw': 226, 'swaspa': 227, 'swathr': 228, 'treswa': 229,
    'truswa': 230, 'tuftit': 231, 'tunswa': 232, 'veery': 233, 'vesspa': 234,
    'vigswa': 235, 'warvir': 236, 'wesblu': 237, 'wesgre': 238, 'weskin': 239,
    'wesmea': 240, 'wessan': 241, 'westan': 242, 'wewpew': 243, 'whbnut': 244,
    'whcspa': 245, 'whfibi': 246, 'whtspa': 247, 'whtswi': 248, 'wilfly': 249,
    'wilsni1': 250, 'wiltur': 251, 'winwre3': 252, 'wlswar': 253, 'wooduc': 254,
    'wooscj2': 255, 'woothr': 256, 'y00475': 257, 'yebfly': 258, 'yebsap': 259,
    'yehbla': 260, 'yelwar': 261, 'yerwar': 262, 'yetvir': 263
}

CODE2BIRD = {v:k for k,v in BIRD_CODE.items()}
```

Audio Preprocessing

Utils

In [ ]:

```
class ReadAudio(object):

    def __init__(
        self
    ):
        pass

    def __call__(self, filename: str) -> Tuple[np.ndarray, int]:
        au, sr = librosa.load(filename, sr=None, dtype=np.float64)
        return au, sr

class PreprocessWaveform(object):

    def __init__(
        self,
        target_sr: int,
        normalize: bool = True,
    ):
        self.target_sr=target_sr
        self.normalize=normalize

    def __call__(self, wave: np.ndarray, sr: int) -> np.ndarray:
        au = librosa.resample(wave, sr, self.target_sr)
        if self.normalize:
            au = librosa.util.normalize(au)
        return au
```

Main Pipeline

In [ ]:

```
class InfernceDataPipelineFileName2Au(object):

    def __init__(
        self,
        target_sr: bool,
        pp_normalize: bool,
    ):
        self.au_reader = ReadAudio()
        self.au_pp = PreprocessWaveform(
            target_sr=target_sr,
            normalize=pp_normalize
        )

    def __call__(
        self,
        au_path: str
    ):
        readed_au, readed_sr = self.au_reader(au_path)
        au = self.au_pp(readed_au, readed_sr)
        pp_sr = self.au_pp.target_sr

        result = {
            'au':au,
            'sr':pp_sr,
        }

        return result
```

Torch Dataset

In [ ]:

```
class Test2StepDataset(torch.utils.data.Dataset):
    def __init__(
        self,
        df: pd.DataFrame,
        root_path: str,
        data_pipeline_austep: object,
        extension: str = '.mp3',
        duration: int = 5
        ):
        self.df = df
        self.data_pipeline_austep = data_pipeline_austep
        self.root_path = root_path
        self.extension = extension
        self.duration = duration

        self.dumped_features = None

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

    def __getitem__(self, idx: int):
        # Extract data from dataframe
        sample = self.df.loc[idx, :]
        site = sample.site
        row_id = sample.row_id
        name = sample.audio_id

        # We have to load new audio if only
        # 1. We get new filename
        # 2. We just started and we do not have loaded audio
        if self.dumped_features is None or self.dumped_features['name'] != name:
            self.dumped_features = {
                'name': name,
                'features': self.data_pipeline_austep(pjoin(self.root_path, name + self.extension))
            }
            print(f'{name} feature loaded')

        if site == "site_3":
            n_secs = len(self.dumped_features['features']['au']) / self.dumped_features['features']['sr']
            sample_rate = self.dumped_features['features']['sr']

            start_sec = 0
            specs = []
            names = []
            sites = []
            row_ids = []

            while n_secs >= start_sec:

                y_batch = self.dumped_features['features']['au']
                # Get duration num of seconds
                y_batch = y_batch[start_sec*sample_rate:(start_sec+self.duration)*sample_rate]
                y_batch = y_batch.astype(np.float32)

                # Do not process audio less then one sec
                if len(y_batch) < MIN_SEC * sample_rate:
                    break

                # Padd if we do not have 5 secs in segment
                if len(y_batch) < sample_rate * self.duration:
                    y_pad = np.zeros(sample_rate * self.duration, dtype=np.float32)
                    y_pad[:len(y_batch)] = y_batch
                    y_batch = y_pad

                y_batch = torch.from_numpy(y_batch)

                start_sec += self.duration

                specs.append(y_batch)
                names.append(name)
                sites.append(sites)
                row_ids.append(row_id)
            return specs, names, sites, row_ids
        else:
            end_seconds = int(sample.seconds)
            start_seconds = int(end_seconds - self.duration)

            sample_rate = self.dumped_features['features']['sr']

            y_batch = self.dumped_features['features']['au']
            y_batch = y_batch[start_seconds*sample_rate:end_seconds*sample_rate]
            y_batch = y_batch.astype(np.float32)

            if len(y_batch) < sample_rate * self.duration:
                y_pad = np.zeros(sample_rate * self.duration, dtype=np.float32)
                y_pad[:len(y_batch)] = y_batch
                y_batch = y_pad

            y_batch = torch.from_numpy(y_batch)

            return y_batch, name, site, row_id
```

Load Dataframe

In [ ]:

```
TEST = Path("../input/birdsong-recognition/test_audio").exists()

if TEST:
    DATA_DIR = Path("../input/birdsong-recognition/")
else:
    # dataset created by @shonenkov, thanks!
    DATA_DIR = Path("../input/birdcall-check/")


test_df = pd.read_csv(DATA_DIR / "test.csv")
test_root = DATA_DIR / "test_audio"


test_df.head()
```

Initialize Dataset

In [ ]:

```
test_dataset = Test2StepDataset(
    df=test_df,
    root_path=test_root,
    data_pipeline_austep=InfernceDataPipelineFileName2Au(
        target_sr=TARGET_SR,
        pp_normalize=PP_NORMALIZE
    ),
    extension='.mp3',
    duration=5
)
```

torchaudio utils

In [ ]:

```
def compute_deltas(
        specgram: torch.Tensor,
        win_length: int = 5,
        mode: str = "replicate"
) -> torch.Tensor:
    r"""Compute delta coefficients of a tensor, usually a spectrogram:

    .. math::
       d_t = \frac{\sum_{n=1}^{\text{N}} n (c_{t+n} - c_{t-n})}{2 \sum_{n=1}^{\text{N}} n^2}

    where :math:`d_t` is the deltas at time :math:`t`,
    :math:`c_t` is the spectrogram coeffcients at time :math:`t`,
    :math:`N` is ``(win_length-1)//2``.

    Args:
        specgram (Tensor): Tensor of audio of dimension (..., freq, time)
        win_length (int, optional): The window length used for computing delta (Default: ``5``)
        mode (str, optional): Mode parameter passed to padding (Default: ``"replicate"``)

    Returns:
        Tensor: Tensor of deltas of dimension (..., freq, time)

    Example
        >>> specgram = torch.randn(1, 40, 1000)
        >>> delta = compute_deltas(specgram)
        >>> delta2 = compute_deltas(delta)
    """
    device = specgram.device
    dtype = specgram.dtype

    # pack batch
    shape = specgram.size()
    specgram = specgram.reshape(1, -1, shape[-1])

    assert win_length >= 3

    n = (win_length - 1) // 2

    # twice sum of integer squared
    denom = n * (n + 1) * (2 * n + 1) / 3

    specgram = torch.nn.functional.pad(specgram, (n, n), mode=mode)

    kernel = torch.arange(-n, n + 1, 1, device=device, dtype=dtype).repeat(specgram.shape[1], 1, 1)

    output = torch.nn.functional.conv1d(specgram, kernel, groups=specgram.shape[1]) / denom

    # unpack batch
    output = output.reshape(shape)

    return output

def make_delta(
    input_tensor: torch.Tensor
):
    input_tensor = input_tensor.transpose(3,2)
    input_tensor = compute_deltas(input_tensor)
    input_tensor = input_tensor.transpose(3,2)
    return input_tensor
```

In [ ]:

```
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=True):
        """Implementation of STFT with Conv1d. The function has the same output
        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)

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

        if freeze_parameters:
            for param in self.parameters():
                param.requires_grad = False

    def forward(self, input):
        """input: (batch_size, data_length)
        Returns:
          real: (batch_size, n_fft // 2 + 1, time_steps)
          imag: (batch_size, n_fft // 2 + 1, time_steps)
        """

        x = input[:, None, :]   # (batch_size, channels_num, data_length)

        if self.center:
            x = F.pad(x, pad=(self.n_fft // 2, self.n_fft // 2), mode=self.pad_mode)

        real = self.conv_real(x)
        imag = self.conv_imag(x)
        # (batch_size, n_fft // 2 + 1, time_steps)

        real = real[:, None, :, :].transpose(2, 3)
        imag = imag[:, None, :, :].transpose(2, 3)
        # (batch_size, 1, time_steps, n_fft // 2 + 1)

        return real, imag


class Spectrogram(nn.Module):
    def __init__(self, n_fft=2048, hop_length=None, win_length=None,
        window='hann', center=True, pad_mode='reflect', power=2.0,
        freeze_parameters=True):
        """Calculate spectrogram using pytorch. The STFT is implemented with
        Conv1d. The function has the same output of librosa.core.stft
        """
        super(Spectrogram, self).__init__()

        self.power = power

        self.stft = STFT(n_fft=n_fft, hop_length=hop_length,
            win_length=win_length, window=window, center=center,
            pad_mode=pad_mode, freeze_parameters=True)

    def forward(self, input):
        """input: (batch_size, 1, time_steps, n_fft // 2 + 1)
        Returns:
          spectrogram: (batch_size, 1, time_steps, n_fft // 2 + 1)
        """

        (real, imag) = self.stft.forward(input)
        # (batch_size, n_fft // 2 + 1, time_steps)

        spectrogram = real ** 2 + imag ** 2

        if self.power == 2.0:
            pass
        else:
            spectrogram = spectrogram ** (self.power / 2.0)

        return spectrogram


class LogmelFilterBank(nn.Module):
    def __init__(self, sr=32000, n_fft=2048, n_mels=64, fmin=50, fmax=14000, is_log=True,
        ref=1.0, amin=1e-10, top_db=80.0, freeze_parameters=True):
        """Calculate logmel spectrogram using pytorch. The mel filter bank is
        the pytorch implementation of as librosa.filters.mel
        """
        super(LogmelFilterBank, self).__init__()

        self.is_log = is_log
        self.ref = ref
        self.amin = amin
        self.top_db = top_db
```

```python
        self.melW = librosa.filters.mel(sr=sr, n_fft=n_fft, n_mels=n_mels,
            fmin=fmin, fmax=fmax).T
        # (n_fft // 2 + 1, mel_bins)

        self.melW = nn.Parameter(torch.Tensor(self.melW))

        if freeze_parameters:
            for param in self.parameters():
                param.requires_grad = False

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

        Output: (batch_size, time_steps, mel_bins)
        """

        # Mel spectrogram
        mel_spectrogram = torch.matmul(input, self.melW)

        # Logmel spectrogram
        if self.is_log:
            output = self.power_to_db(mel_spectrogram)
        else:
            output = mel_spectrogram

        return output


    def power_to_db(self, input):
        """Power to db, this function is the pytorch implementation of
        librosa.core.power_to_lb
        """
        ref_value = self.ref
        log_spec = 10.0 * torch.log10(torch.clamp(input, min=self.amin, max=np.inf))
        log_spec -= 10.0 * np.log10(np.maximum(self.amin, ref_value))

        if self.top_db is not None:
            if self.top_db < 0:
                raise ValueError('top_db must be non-negative')
            for i in range(log_spec.shape[0]):
                log_spec[i] = torch.clamp(log_spec[i], min=log_spec[i].max().item() - self.top_db, max=np.inf)

        return log_spec

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__()

        self.time_dropper = DropStripes(dim=2, drop_width=time_drop_width,
            stripes_num=time_stripes_num)

        self.freq_dropper = DropStripes(dim=3, drop_width=freq_drop_width,
            stripes_num=freq_stripes_num)

    def forward(self, input):
        x = self.time_dropper(input)
        x = self.freq_dropper(x)
        return x

class Loudness(nn.Module):
    def __init__(self, sr, n_fft, min_db, device):
        super().__init__()
        self.min_db = min_db
        freqs = librosa.fft_frequencies(
            sr=sr, n_fft=n_fft
        )
        self.a_weighting = torch.nn.Parameter(
            data=torch.from_numpy(librosa.A_weighting(freqs + 1e-10)),
            requires_grad=False
        )
        self.to(device)

    def forward(self, spec):
        power_db = torch.log10(spec**0.5 + 1e-10)

        loudness = power_db + self.a_weighting

        #loudness -= 10 * torch.log10(spec)
        loudness -= 20.7

        loudness = torch.clamp(loudness, min=-self.min_db)

        # Average over frequency bins.
        loudness = torch.mean(loudness, axis=-1).float()
        return loudness
```

Resnext initialize functions

In [ ]:

```
def _resnext(block, layers, pretrained, progress, **kwargs):
    return ResNet(block, layers, **kwargs)

def resnext50_32x4d_swsl(progress=True, **kwargs):
    """Constructs a semi-weakly supervised ResNeXt-50 32x4 model pre-trained on 1B weakly supervised
       image dataset and finetuned on ImageNet.
       `"Billion-scale Semi-Supervised Learning for Image Classification" <https://arxiv.org/abs/1905.00546>`_
    Args:
        progress (bool): If True, displays a progress bar of the download to stderr.
    """
    kwargs['groups'] = 32
    kwargs['width_per_group'] = 4
    return _resnext(Bottleneck, [3, 4, 6, 3], True, progress, **kwargs)

def resnext101_32x4d_swsl(progress=True, **kwargs):
    """Constructs a semi-weakly supervised ResNeXt-101 32x4 model pre-trained on 1B weakly supervised
       image dataset and finetuned on ImageNet.
       `"Billion-scale Semi-Supervised Learning for Image Classification" <https://arxiv.org/abs/1905.00546>`_
    Args:
        progress (bool): If True, displays a progress bar of the download to stderr.
    """
    kwargs['groups'] = 32
    kwargs['width_per_group'] = 4
    return _resnext(Bottleneck, [3, 4, 23, 3], True, progress, **kwargs)
```

BIG BIG Model class

In [ ]:

```
EFFNETB6_EMB_DIM = 2304
EFFNETB5_EMB_DIM = 2048
EFFNETB4_EMB_DIM = 1792
EFFNETB3_EMB_DIM = 1536
EFFNETB1_EMB_DIM = 1280
RESNEST50_FAST_EMB_DIM = 2048
RESNEXT50_EMB_DIM = 2048
RESNEXT101_EMB_DIM = 2048

EPS = 1e-6

class SpectralEffnet(nn.Module):
    def __init__(
        self,
        # Spectral Config ==Start==
        sample_rate: int,
        window_size: int,
        hop_size: int,
        mel_bins: int,
        fmin: int,
        fmax: int,
        top_db: float,
        # Spectral Config ==End==
        # Model Config ==Start==
        classes_num: int,
        encoder_type: str,
        hidden_dims: int,
        first_dropout_rate: float,
        second_dropout_rate: float,
        use_pretrained_encoder: bool,
        classifier_type: str,
        # Model Config ==End==
        # Feature Extraction Config ==Start==
        use_spectral_cutout: bool,
        spec_aggreagation: str,
        use_loudness: bool,
        use_spectral_centroid: bool,
        # Feature Extraction Config ==End==
        # Other Config ==Start==
        device: str,
        use_sigmoid: bool = True,
        # Other Config ==End==
    ):
        super().__init__()

        window = 'hann'
        center = True
        pad_mode = 'reflect'
        ref = 1.0
        amin = 1e-10
        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)

        if use_loudness:
            self.loudness_bn = nn.BatchNorm1d(1)
            self.loudness_extractor = Loudness(
                sr=sample_rate,
                n_fft=window_size,
                min_db=120,
                device=device
            )
        if use_spectral_centroid:
            self.spectral_centroid_bn = nn.BatchNorm1d(1)

        self.bn0 = nn.BatchNorm2d(mel_bins)

        if spec_aggreagation in ['conv1', 'repeat3', 'deltas', 'conv3', 'time_freq_encoding']:
            self.spec_aggreagation = spec_aggreagation
        else:
            raise ValueError('Invalid spec_aggreagation')

        if encoder_type == 'effnet1':
            self.encoder = geffnet.tf_efficientnet_b1_ns(pretrained=use_pretrained_encoder)
            self.encoder.classifier = nn.Identity()
            if self.spec_aggreagation == 'conv1':
                self.encoder.conv_stem = Conv2dSame(
                    in_channels=1,
                    out_channels=self.encoder.conv_stem.out_channels,
                    kernel_size=self.encoder.conv_stem.kernel_size,
                    stride=self.encoder.conv_stem.stride,
                    bias=self.encoder.conv_stem.bias
                    )
            nn_embed_size = EFFNETB1_EMB_DIM
        elif encoder_type == 'effnet3':
            self.encoder = geffnet.tf_efficientnet_b3_ns(pretrained=use_pretrained_encoder)
            self.encoder.classifier = nn.Identity()
            if self.spec_aggreagation == 'conv1':
                self.encoder.conv_stem = Conv2dSame(
                    in_channels=1,
                    out_channels=self.encoder.conv_stem.out_channels,
                    kernel_size=self.encoder.conv_stem.kernel_size,
                    stride=self.encoder.conv_stem.stride,
                    bias=self.encoder.conv_stem.bias
                    )
            nn_embed_size = EFFNETB3_EMB_DIM
        elif encoder_type == 'effnet4':
            self.encoder = geffnet.tf_efficientnet_b4_ns(pretrained=use_pretrained_encoder)
            self.encoder.classifier = nn.Identity()
            if self.spec_aggreagation == 'conv1':
                self.encoder.conv_stem = Conv2dSame(
                    in_channels=1,
                    out_channels=self.encoder.conv_stem.out_channels,
                    kernel_size=self.encoder.conv_stem.kernel_size,
                    stride=self.encoder.conv_stem.stride,
                    bias=self.encoder.conv_stem.bias
                    )
            nn_embed_size = EFFNETB4_EMB_DIM
        elif encoder_type == 'effnet5':
            self.encoder = geffnet.tf_efficientnet_b5_ns(pretrained=use_pretrained_encoder)
            self.encoder.classifier = nn.Identity()
            if self.spec_aggreagation == 'conv1':
                self.encoder.conv_stem = Conv2dSame(
                    in_channels=1,
                    out_channels=self.encoder.conv_stem.out_channels,
                    kernel_size=self.encoder.conv_stem.kernel_size,
                    stride=self.encoder.conv_stem.stride,
                    bias=self.encoder.conv_stem.bias
                    )
            nn_embed_size = EFFNETB5_EMB_DIM
        elif encoder_type == 'effnet6':
            self.encoder = geffnet.tf_efficientnet_b6_ns(pretrained=use_pretrained_encoder)
            self.encoder.classifier = nn.Identity()
            if self.spec_aggreagation == 'conv1':
                self.encoder.conv_stem = Conv2dSame(
                    in_channels=1,
                    out_channels=self.encoder.conv_stem.out_channels,
                    kernel_size=self.encoder.conv_stem.kernel_size,
                    stride=self.encoder.conv_stem.stride,
                    bias=self.encoder.conv_stem.bias
                    )
            nn_embed_size = EFFNETB6_EMB_DIM
        elif encoder_type == 'resnest50_fast_1s1x64d':
            if self.spec_aggreagation == 'conv1':
                raise ValueError('Invalid spec_aggreagation for this encoder')
            self.encoder = resnest50_fast_1s1x64d(pretrained=use_pretrained_encoder)
            self.encoder.fc = nn.Identity()
            nn_embed_size = RESNEST50_FAST_EMB_DIM
        elif encoder_type == 'resnext50_32x4d_swsl':
            self.encoder = resnext50_32x4d_swsl()
            if use_pretrained_encoder:
                self.encoder.load_state_dict(
                    torch.hub.load('facebookresearch/semi-supervised-ImageNet1K-models', 'resnext50_32x4d_swsl').
                )
            self.encoder.fc = nn.Identity()
            if self.spec_aggreagation == 'conv1':
                self.encoder.conv1 = nn.Conv2d(
                    in_channels=1,
                    out_channels=self.encoder.conv1.out_channels,
                    kernel_size=self.encoder.conv1.kernel_size,
                    stride=self.encoder.conv1.stride,
                    bias=self.encoder.conv1.bias
                    )
            nn_embed_size = RESNEXT50_EMB_DIM
        elif encoder_type == 'resnext101_32x4d_swsl':
            self.encoder = resnext101_32x4d_swsl()
            if use_pretrained_encoder:
                self.encoder.load_state_dict(
                    torch.hub.load('facebookresearch/semi-supervised-ImageNet1K-models', 'resnext101_32x4d_swsl')
                )
            self.encoder.fc = nn.Identity()
            if self.spec_aggreagation == 'conv1':
                self.encoder.conv1 = nn.Conv2d(
                    in_channels=1,
                    out_channels=self.encoder.conv1.out_channels,
                    kernel_size=self.encoder.conv1.kernel_size,
                    stride=self.encoder.conv1.stride,
                    bias=self.encoder.conv1.bias
                    )
            nn_embed_size = RESNEXT101_EMB_DIM
        else:
            raise ValueError(f'{encoder_type} is invalid model_type')


        if classifier_type == 'relu':
            self.classifier = nn.Sequential(
                nn.Linear(nn_embed_size, hidden_dims), nn.ReLU(), nn.Dropout(p=first_dropout_rate),
                nn.Linear(hidden_dims, hidden_dims), nn.ReLU(), nn.Dropout(p=second_dropout_rate),
                nn.Linear(hidden_dims, classes_num)
            )
        elif classifier_type == 'elu':
            self.classifier = nn.Sequential(
                nn.Dropout(first_dropout_rate),
                nn.Linear(nn_embed_size, hidden_dims),
                nn.ELU(),
                nn.Dropout(second_dropout_rate),
                nn.Linear(hidden_dims, classes_num)
            )
        elif classifier_type == 'dima': # inspired by https://github.com/ex4sperans/freesound-classification
            self.classifier = nn.Sequential(
                nn.BatchNorm1d(nn_embed_size),
                nn.Linear(nn_embed_size, hidden_dims),
                nn.BatchNorm1d(hidden_dims),
                nn.PReLU(hidden_dims),
                nn.Dropout(p=second_dropout_rate),
                nn.Linear(hidden_dims, classes_num)
            )
        elif classifier_type == 'prelu':
            self.classifier = nn.Sequential(
                nn.Dropout(first_dropout_rate),
                nn.Linear(nn_embed_size, hidden_dims),
                nn.PReLU(hidden_dims),
                nn.Dropout(p=second_dropout_rate),
                nn.Linear(hidden_dims, classes_num)
            )
        elif classifier_type == 'multiscale_relu':
            self.big_dropout = nn.Dropout(p=0.5)
            self.classifier = nn.Sequential(
                nn.Linear(nn_embed_size, hidden_dims), nn.ReLU(), nn.Dropout(p=first_dropout_rate),
                nn.Linear(hidden_dims, hidden_dims), nn.ReLU(), nn.Dropout(p=second_dropout_rate),
                nn.Linear(hidden_dims, classes_num)
            )
        else:
            raise ValueError("Invalid classifier_type")

        # Classifier type
        self.classifier_type = classifier_type
        # Augmentations
        self.use_spectral_cutout = use_spectral_cutout
        # Final activation
        self.use_sigmoid = use_sigmoid
        # Additional features
        self.use_spectral_centroid = use_spectral_centroid
        self.use_loudness = use_loudness
        # Some additional stuff
        self.encoder_type = encoder_type
        self.device = device
        self.mel_bins = mel_bins
        self.to(self.device)

    def _add_frequency_encoding(self, x):
        n, d, h, w = x.size()

        vertical = torch.linspace(-1, 1, w, device=x.device).view(1, 1, 1, -1)
        vertical = vertical.repeat(n, 1, h, 1)

        return vertical
    def _add_time_encoding(self, x):
        n, d, h, w = x.size()

        horizontal = torch.linspace(-1, 1, h, device=x.device).view(1, 1, -1, 1)
        horizontal = horizontal.repeat(n, 1, 1, w)

        return horizontal

    def preprocess(self, input):
        x = self.spectrogram_extractor(input)  # (batch_size, 1, time_steps, freq_bins)

        additional_features = []
        if self.use_loudness:
            loudness = self.loudness_extractor(x)
            loudness = self.loudness_bn(loudness)
            loudness = loudness.unsqueeze(-1)
            loudness = loudness.repeat(1,1,1,self.mel_bins)
            additional_features.append(loudness)
        if self.use_spectral_centroid:
            spectral_centroid = x.mean(-1)
            spectral_centroid = self.spectral_centroid_bn(spectral_centroid)
            spectral_centroid = spectral_centroid.unsqueeze(-1)
            spectral_centroid = spectral_centroid.repeat(1,1,1,self.mel_bins)
            additional_features.append(spectral_centroid)

        x = self.logmel_extractor(x)  # (batch_size, 1, time_steps, mel_bins)

        if self.training and self.use_spectral_cutout:
            x = self.spec_augmenter(x)

        frames_num = x.shape[2]

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

        if len(additional_features) > 0:
            additional_features.append(x)
            x = torch.cat(additional_features, dim=1)

        if self.spec_aggreagation == 'repeat3':
            x = torch.cat([x,x,x], dim=1)
        elif self.spec_aggreagation == 'deltas':
            delta_1 = make_delta(x)
            delta_2 = make_delta(delta_1)
            x = torch.cat([x,delta_1,delta_2], dim=1)
        elif self.spec_aggreagation == 'time_freq_encoding':
            freq_encode = self._add_frequency_encoding(x)
            time_encode = self._add_time_encoding(x)
            x = torch.cat([x, freq_encode, time_encode], dim=1)
        elif self.spec_aggreagation in ['conv1','conv3']:
            pass

        return x, frames_num


    def forward(self, input):
        """
        Input: (batch_size, data_length)
        """
        # Output shape (batch size, channels, time, frequency)
        x, _ = self.preprocess(input)

        # Output shape (batch size, channels)
        x = self.encoder(x)

        if self.classifier_type == 'multiscale_relu':
            logits = torch.mean(torch.stack([self.classifier(self.big_dropout(x)) for _ in range(5)],dim=0),dim=0
        else:
            logits = self.classifier(x)

        if self.use_sigmoid:
            return torch.sigmoid(logits)
        else:
            return logits
```

Model utils

In [ ]:

```
def load_chkp(
    path: str,
    nn_model: nn.Module
):
    chckp = torch.load(path, map_location='cpu')
    nn_model.load_state_dict(chckp)
    return nn_model


def initalize_spectral_model(
    model_config: Mapping[str, Any],
    chkp_path: str,
    m_device: str
):
    temp_model = SpectralEffnet(
        **model_config,
        device=m_device
    )
    temp_model = load_chkp(chkp_path, temp_model)
    temp_model.eval()
    return temp_model
```

ALL My Models

All models are taken from 5 folds, so each experiment contains bunch of 5 models.

Each model is created by SWA(simple averaging of weight matrices from 3 best chekpoints taken for `f1_test_median`)

In [ ]:

```
NEW_EFFNET3 = {
    "sample_rate": TARGET_SR,
    "window_size": 2048,
    "hop_size": 256,
    "mel_bins": 128,
    "fmin": 20,
    "fmax": 16000,
    "top_db":80.0,
    "classes_num": len(BIRD_CODE),
    "encoder_type": 'effnet3',
    "hidden_dims": 1024,
    "first_dropout_rate": 0.2,
    "second_dropout_rate": 0.2,
    "use_spectral_cutout": False,
    "spec_aggreagation": "deltas",
    "use_pretrained_encoder": False,
    "use_sigmoid": True,
    "classifier_type": "relu",
    "use_loudness": False,
    "use_spectral_centroid": False,
}
NEW_EFFNET3_MD = {
    "sample_rate": TARGET_SR,
    "window_size": 2048,
    "hop_size": 256,
    "mel_bins": 128,
    "fmin": 20,
    "fmax": 16000,
    "top_db":80.0,
    "classes_num": len(BIRD_CODE),
    "encoder_type": 'effnet3',
    "hidden_dims": 1024,
    "first_dropout_rate": 0.2,
    "second_dropout_rate": 0.2,
    "use_spectral_cutout": False,
    "spec_aggreagation": "deltas",
    "use_pretrained_encoder": False,
    "use_sigmoid": True,
    "classifier_type": "multiscale_relu",
    "use_loudness": False,
    "use_spectral_centroid": False,
}
NEW_EFFNET4 = {
    "sample_rate": TARGET_SR,
    "window_size": 2048,
    "hop_size": 256,
    "mel_bins": 128,
    "fmin": 20,
    "fmax": 16000,
    "top_db":80.0,
    "classes_num": len(BIRD_CODE),
    "encoder_type": 'effnet4',
    "hidden_dims": 1024,
    "first_dropout_rate": 0.2,
    "second_dropout_rate": 0.2,
    "use_spectral_cutout": False,
    "spec_aggreagation": "deltas",
    "use_pretrained_encoder": False,
    "use_sigmoid": True,
    "classifier_type": "relu",
    "use_loudness": False,
    "use_spectral_centroid": False,
```

```python
}
NEW_EFFNET4_MD = {
    "sample_rate": TARGET_SR,
    "window_size": 2048,
    "hop_size": 256,
    "mel_bins": 128,
    "fmin": 20,
    "fmax": 16000,
    "top_db":80.0,
    "classes_num": len(BIRD_CODE),
    "encoder_type": 'effnet4',
    "hidden_dims": 1024,
    "first_dropout_rate": 0.2,
    "second_dropout_rate": 0.2,
    "use_spectral_cutout": False,
    "spec_aggreagation": "deltas",
    "use_pretrained_encoder": False,
    "use_sigmoid": True,
    "classifier_type": "multiscale_relu",
    "use_loudness": False,
    "use_spectral_centroid": False,
}
NEW_EFFNET5_MD = {
    "sample_rate": TARGET_SR,
    "window_size": 2048,
    "hop_size": 256,
    "mel_bins": 128,
    "fmin": 20,
    "fmax": 16000,
    "top_db":80.0,
    "classes_num": len(BIRD_CODE),
    "encoder_type": 'effnet5',
    "hidden_dims": 1024,
    "first_dropout_rate": 0.2,
    "second_dropout_rate": 0.2,
    "use_spectral_cutout": False,
    "spec_aggreagation": "deltas",
    "use_pretrained_encoder": False,
    "use_sigmoid": True,
    "classifier_type": "multiscale_relu",
    "use_loudness": False,
    "use_spectral_centroid": False,
}
NEW_EFFNET3_MD_TFE = {
    "sample_rate": TARGET_SR,
    "window_size": 2048,
    "hop_size": 256,
    "mel_bins": 128,
    "fmin": 20,
    "fmax": 16000,
    "top_db":80.0,
    "classes_num": len(BIRD_CODE),
    "encoder_type": 'effnet3',
    "hidden_dims": 1024,
    "first_dropout_rate": 0.2,
    "second_dropout_rate": 0.2,
    "use_spectral_cutout": False,
    "spec_aggreagation": "time_freq_encoding",
    "use_pretrained_encoder": False,
    "use_sigmoid": True,
    "classifier_type": "multiscale_relu",
    "use_loudness": False,
    "use_spectral_centroid": False,
}


models = [
# effnet3_multiscalereluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_seconda
# Public score: 0.613 (thresh - 0.4) and 0.612 (thresh - 0.5)
    initalize_spectral_model(NEW_EFFNET3_MD, '../input/cornell-birds-models/effnet3_multiscalereluclas_plato_delt
    initalize_spectral_model(NEW_EFFNET3_MD, '../input/cornell-birds-models/effnet3_multiscalereluclas_plato_delt
    initalize_spectral_model(NEW_EFFNET3_MD, '../input/cornell-birds-models/effnet3_multiscalereluclas_plato_delt
    initalize_spectral_model(NEW_EFFNET3_MD, '../input/cornell-birds-models/effnet3_multiscalereluclas_plato_delt
    initalize_spectral_model(NEW_EFFNET3_MD, '../input/cornell-birds-models/effnet3_multiscalereluclas_plato_delt
# effnet4_reluclas_plato_deltas_50bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_m
# Public score: 0.61 (thresh - 0.4), 0.605 (thresh - 0.5)
    initalize_spectral_model(NEW_EFFNET4, '../input/cornell-birds-models/effnet4_reluclas_plato_deltas_50bs_001l
    initalize_spectral_model(NEW_EFFNET4, '../input/cornell-birds-models/effnet4_reluclas_plato_deltas_50bs_001l
    initalize_spectral_model(NEW_EFFNET4, '../input/cornell-birds-models/effnet4_reluclas_plato_deltas_50bs_001l
    initalize_spectral_model(NEW_EFFNET4, '../input/cornell-birds-models/effnet4_reluclas_plato_deltas_50bs_001l
    initalize_spectral_model(NEW_EFFNET4, '../input/cornell-birds-models/effnet4_reluclas_plato_deltas_50bs_001l
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_m
# Public score: 0.608 (thresh - 0.4), 0.607 (thresh - 0.5)
    initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001l
    initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001l
    initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001l
    initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001l
    initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001l
# effnet4_multiscalereluclas_plato_deltas_50bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_seconda
# Public score: 0.601 (thresh - 0.4)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
#  initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
#  initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
#  initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
#  initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_mi
# Public score: 0.599 (thresh - 0.5)
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_mi
# Public score: 0.601 (thresh - 0.5)
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_mi
# Public score: 0.606 (thresh - 0.5)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
# effnet5_multiscalereluclas_plato_deltas_32bs_2acum_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_se
# Public score: 0.607 (thresh - 0.5)
  initalize_spectral_model(NEW_EFFNET5_MD, '../input/cornell-birds-models/effnet5_multiscalereluclas_plato_delt
  initalize_spectral_model(NEW_EFFNET5_MD, '../input/cornell-birds-models/effnet5_multiscalereluclas_plato_delt
  initalize_spectral_model(NEW_EFFNET5_MD, '../input/cornell-birds-models/effnet5_multiscalereluclas_plato_delt
  initalize_spectral_model(NEW_EFFNET5_MD, '../input/cornell-birds-models/effnet5_multiscalereluclas_plato_delt
  initalize_spectral_model(NEW_EFFNET5_MD, '../input/cornell-birds-models/effnet5_multiscalereluclas_plato_delt
# effnet3_multiscalereluclas_plato_timefreqencode_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_
# Public score: 0.595 (thresh - 0.5)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET3_MD_TFE, '../input/cornell-birds-models/effnet3_multiscalereluclas_plat
#  initalize_spectral_model(NEW_EFFNET3_MD_TFE, '../input/cornell-birds-models/effnet3_multiscalereluclas_plat
#  initalize_spectral_model(NEW_EFFNET3_MD_TFE, '../input/cornell-birds-models/effnet3_multiscalereluclas_plat
#  initalize_spectral_model(NEW_EFFNET3_MD_TFE, '../input/cornell-birds-models/effnet3_multiscalereluclas_plat
#  initalize_spectral_model(NEW_EFFNET3_MD_TFE, '../input/cornell-birds-models/effnet3_multiscalereluclas_plat
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_mi
# Public score: 0.589 (thresh - 0.5)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_mi
# Public score: 0.598 (thresh - 0.5)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_ls
# Public score: 0.596 (thresh - 0.5)
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_001lr
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_ls
# Public score: 0.593 (thresh - 0.5)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels
# Public score: 0.582 (thresh - 0.5)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
# effnet3_reluclas_plato_deltas_64bs_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugs_secondarylabels_ex
# Public score: 0.601 (thresh - 0.5)
# Is not used in best Blend
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#  initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#     initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
#     initalize_spectral_model(NEW_EFFNET3, '../input/cornell-birds-models/effnet3_reluclas_plato_deltas_64bs_00
# effnet4_multiscalereluclas_plato_deltas_32bs_2acum_001lr_biggerfft_trackmap_energytrimming_topdb80_firstaugsle
# Public score: not commited
# Is not used in best Blend
#     initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
#     initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
#     initalize_spectral_model(NEW_EFFNET4_MD, '../input/cornell-birds-models/effnet4_multiscalereluclas_plato_de
]
```

In [ ]:

```
print(f"Models in Blend: {len(models)}")
```

Predict Loop

In [ ]:

```
batch = []
names = []
probs = []
for i in tqdm(range(len(test_dataset))):
    sample_spec, _, _, sample_id = test_dataset[i]

    # If we have site3 we get list of samples
    # If we have site1 or site2 - only one sample
    if isinstance(sample_spec, list):
        batch += sample_spec
    else:
        batch.append(sample_spec)

    if isinstance(sample_id, list):
        names += sample_id
    else:
        names.append(sample_id)

    if len(batch) >= BATCH_SIZE or i == (len(test_dataset) - 1):
        with torch.no_grad():
            # If site3 produced very big batch we decompose it
            # in smaller ones
            if len(batch) > BATCH_SIZE + 1:
                n_steps = math.ceil(len(batch) / BATCH_SIZE)
                for step in range(n_steps):
                    small_batch = batch[step*BATCH_SIZE:(step+1)*BATCH_SIZE]
                    small_batch = torch.stack(small_batch).to(DEVICE).float()
                    # Or models are blend with simple Mean
                    batch_probs = sum(m(small_batch).detach().cpu() for m in models) / len(models)
                    batch_probs = batch_probs.numpy()
                    probs.append(batch_probs)
            else:
                batch = torch.stack(batch).to(DEVICE).float()
                # Or models are blend with simple Mean
                batch_probs = sum(m(batch).detach().cpu() for m in models) / len(models)
                batch_probs = batch_probs.numpy()
                probs.append(batch_probs)

        batch = []
```

Create final submission DF

In [ ]:

```
# Concatanate batches
probs = np.concatenate(probs)

# Create DataFrame
result_df = pd.DataFrame({
    'row_id':names,
    'birds':[probs[i] for i in range(probs.shape[0])]
})

# Aggregate predictions from site3 by Max
result_df = result_df.groupby('row_id')['birds'].apply(lambda x: np.stack(x).max(0)).reset_index()

result_df
```

In [ ]:

```
def probs2names(
    probs_array: np.ndarray,
    threshold: int = 0.5,
    max_birds_to_take: Optional[int] = None,
    idx2bird_mapping: Mapping[int, str] = CODE2BIRD
):
    accepted_indices = np.where(probs_array > threshold)[0]
    if max_birds_to_take is not None:
        accepted_probs = probs_array[accepted_indices]
        accepted_probs_indices = np.argsort(-accepted_probs)
        accepted_indices = accepted_indices[accepted_probs_indices]
        accepted_indices = accepted_indices[:max_birds_to_take]

    if len(accepted_indices) == 0:
        return 'nocall'
    else:
        return ' '.join([idx2bird_mapping[idx] for idx in  accepted_indices])
```

In [ ]:

```
result_df['birds'] = result_df['birds'].apply(lambda x: probs2names(
    x,
    threshold=SIGMOID_THRESH,
    max_birds_to_take=MAX_BIRDS
))
```

In [ ]:

```
submission = test_df.merge(result_df, on='row_id', how='left')[['row_id','birds']]
submission
```

In [ ]:

```
submission['birds'].value_counts(normalize=True)
```

In [ ]:

```
submission.to_csv("submission.csv", index=False)
```

Train Tips

Augmentations

Great thanks for audiomentations package

Gain

Taken from https://github.com/iver56/audiomentations

```
class Gain(audiomentations.BasicTransform):
    """
    Multiply the audio by a random amplitude factor to reduce or increase the volume. This
    technique can help a model become somewhat invariant to the overall gain of the input audio.
    Warning: This transform can return samples outside the [-1, 1] range, which may lead to
    clipping or wrap distortion, depending on what you do with the audio in a later stage.
    See also https://en.wikipedia.org/wiki/Clipping_(audio)#Digital_clipping
    """

    def __init__(self, min_gain_in_db=-12, max_gain_in_db=12, p=0.5):
        """
        :param p:
        """
        super().__init__(p)
        assert min_gain_in_db <= max_gain_in_db
        self.min_gain_in_db = min_gain_in_db
        self.max_gain_in_db = max_gain_in_db

    def randomize_parameters(self, samples, sample_rate):
        super().randomize_parameters(samples, sample_rate)
        if self.parameters["should_apply"]:
            self.parameters["amplitude_ratio"] = convert_decibels_to_amplitude_ratio(
                random.uniform(self.min_gain_in_db, self.max_gain_in_db)
            )

    def apply(self, samples, sample_rate):
        return samples * self.parameters["amplitude_ratio"]
```

Background Noise

```
class SpecifiedNoise(audiomentations.BasicTransform):

    def __init__(
        self,
        noise_folder_path: str,
        p: int = 0.5,
        allways_apply: bool = False,
        low_alpha: float = 0.0,
        high_alpha: float = 1.0
    ):
        super().__init__(p)
        filenames = glob(pjoin(noise_folder_path, '*.wav'))
        self.noises = [librosa.load(noise_path, sr=None)[0] for noise_path in filenames]
        self.noises = [librosa.util.normalize(noise) for noise in self.noises]
        self.p = p
        self.allways_apply = allways_apply
        self.low_alpha = low_alpha
        self.high_alpha = high_alpha

    def apply(self, au, sr):
        if np.random.binomial(n=1, p=self.p) or self.allways_apply:
            alpha = np.random.uniform(low=self.low_alpha, high=self.high_alpha)
            noise = self.noises[np.random.randint(low=0, high=len(self.noises))]
            au = au*(1 - alpha) + noise * alpha

        return au
```

LowFrequency CutOff

```
class LowFrequencyMask(audiomentations.BasicTransform):

    def __init__(
        self,
        p: int = 0.5,
        allways_apply: bool = False,
        max_cutoff: float = 5,
        min_cutoff: float = 4
    ):
        super().__init__(p)
        self.p = p
        self.allways_apply = allways_apply
        self.max_cutoff = max_cutoff
        self.min_cutoff = min_cutoff

    def apply(self, au, sr):
        if np.random.binomial(n=1, p=self.p) or self.allways_apply:
            cutoff_value = np.random.uniform(low=self.min_cutoff, high=self.max_cutoff)
            au = butter_lowpass_filter(au, cutoff=cutoff_value, fs=sr / 1000)

        return au
```

All Pipeline

`noisy_samples` you can find in this dataset

```
audiomentations.Compose([
                        Gain(p=0.5),
                        SpecifiedNoise('/ssd_data/birdsong_recognition/noisy_samples', low_alpha=0.5,
high_alpha=0.8, p=0.1),

audiomentations.AddBackgroundNoise('/ssd_data/birdsong_recognition/noisy_samples',
min_snr_in_db=0.001, max_snr_in_db=2, p=0.75),
                        LowFrequencyMask(min_cutoff=5, max_cutoff=7, p=0.75)
])
```

Mixup

I have used OR Mixup, that was proposed by Dmytro Danevskyi here

I have used high mixup probability - 75%

Here is full loss function, which includes Mixup and Label Smoothing. But Label Smoothing did not work for me in all cases

In [ ]:

```
import torch
import torch.nn as nn
import numpy as np

class StrongBCEwithLogitsMixUp(nn.Module):

    def __init__(
        self,
        n_classes: int,
        use_onehot_target: bool = True,
        label_smoothing_coef: float = 0.0,
        mixup_p: float = 0.5,
        mixup_type: str = 'wave_sum'
    ):
        super().__init__()
        self.n_classes = n_classes
        self.use_onehot_target = use_onehot_target
        self.label_smoothing_coef = label_smoothing_coef

        self.mixup_p = mixup_p

        if mixup_type not in ['wave_sum']:
            raise ValueError("Wrong mixup_type")
        self.mixup_type = mixup_type

        self.loss_f = nn.BCEWithLogitsLoss()

    def _sum_mixup(self, inputs, targets):
        indices = torch.randperm(inputs.size(0))
        shuffled_inputs = inputs[indices]
        shuffled_targets = targets[indices]

        inputs = inputs + shuffled_inputs
        targets = targets + shuffled_targets

        targets = targets.clamp(min=0, max=1.0)
        for i in range(inputs.shape[0]):
            inputs[i,:] = inputs[i,:] / inputs[i,:].abs().max()

        return inputs, targets

    def forward(self, batch, model):

        wave = batch['waveform']
        target = batch['targets']
        if self.use_onehot_target:
            one_hot_target = torch.nn.functional.one_hot(target, self.n_classes)
            primary_target = target
        else:
            one_hot_target = target
            primary_target = batch['primary_label']

        if model.training and np.random.binomial(n=1,p=self.mixup_p):
            if self.mixup_type == 'wave_sum':
                wave, one_hot_target = self._sum_mixup(wave, one_hot_target)

        logits = model(wave)

        # Handle output for clipwise/framewise model
        if isinstance(logits, dict):
            logits = logits['clipwise_output']

        if self.label_smoothing_coef > 0 and model.training:
            one_hot_target = torch.abs(one_hot_target.float() - self.label_smoothing_coef)
        else:
            one_hot_target = one_hot_target.float()

        loss = self.loss_f(logits, one_hot_target)

        losses = {
            "loss":loss
        }
        outputs = {
            "logits":logits
        }
        inputs = {
            "targets":primary_target,
            "all_targets": target if not self.use_onehot_target else None
        }

        return losses, inputs, outputs
```

Multi-Sample Dropout

Originally proposed in this paper and used by winners of QA competition

```
self.big_dropout = nn.Dropout(p=0.5)
self.classifier = nn.Sequential(
    nn.Linear(nn_embed_size, hidden_dims), nn.ReLU(), nn.Dropout(p=first_dropout_rate),
    nn.Linear(hidden_dims, hidden_dims), nn.ReLU(), nn.Dropout(p=second_dropout_rate),
    nn.Linear(hidden_dims, classes_num)
)
logits = torch.mean(torch.stack([self.classifier(self.big_dropout(x)) for _ in range(5)],dim=0),dim=0)
```

Energy trimming

I have created RMS feature with window size of 5 seconds and step size of 1 second. Then normalized it in order to use as probability distribution for sampling start second of the audio

```
def compute_normalized_energy(
    wave: np.array
):
    wave = wave / np.abs(wave).max()
    return np.power(wave,2)

def compute_sampling_distribution(
    feature: np.ndarray,
    hop_size: int,
    window_size: int,
    probs_comp: str = 'softmax'
):
    n_steps = max(math.ceil((len(feature) - window_size) / hop_size),1)
    if probs_comp == 'softmax':
        areas = [feature[hop_size*i:window_size + hop_size*i].sum() for i in range(n_steps)]
        probs = softmax(areas)
    elif probs_comp == 'uniform':
        areas = [feature[hop_size*i:window_size + hop_size*i].sum() for i in range(n_steps)]
        probs = np.array(areas) / sum(areas)
    elif probs_comp == 'rm_softmax':
        areas = [feature[hop_size*i:window_size + hop_size*i].mean() for i in range(n_steps)]
        probs = softmax(np.power(np.array(areas), 0.5))
    else:
        raise ValueError('Invalid probs_comp')
    return probs

h_s = sr * 1
w_s = sr * 5

t_pobs = compute_sampling_distribution(
    feature=compute_normalized_energy(au),
    hop_size=h_s,
    window_size=w_s,
    probs_comp='uniform'
)

start = np.random.choice(len(t_pobs),size=1, p=t_pobs)[0]
segment = au[start*h_s:start*h_s + w_s ]
```

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js