My partner will be describing the psuedo labeling generation procedure, performance of different model architectures and loss functions in more detail. Here are some points that i found to be important.

1. Catastrophic forgetting in neural networks There is imbalance in the distribution of bird species, for ex: species 3 occurs very frequently. During training I found that initially model learns to classify species 3 and as the training proceeds it starts "forgetting". The confidence for species 3 goes on decreasing which negatively impacts the lb score. So, we need to make sure that other species are learnt without forgetting species 3. I found that recall rate for species 3 can be improved by setting pos_weight in BCELoss. You may find this paper interesting if you are more curious: https://arxiv.org/pdf/1612.00796.pdf (especially section 2.1)
2. Augmenting other datasets\ Not all parts of the audio are occupied by bird species. I replaced these unoccupied parts with bird songs from cornell.
3. Misc
    - Validation scheme should be similar to test scheme. For ex: If you feed 5s chunks during test and then take max, the same thing should be done during validation also.
    - I found Click Noise Augmentation to be very useful (https://librosa.org/doc/0.8.0/generated/librosa.clicks.html)
    - Using pretrained weights (imagenet/cornell) can help to converge much faster.
    - Model Averaging seems to always lead to better generalization.
    - 5s crops seems to perform slightly better than 10s crops

In [1]:

```
!pip install resnest > /dev/null
!pip install colorednoise > /dev/null
```

```
WARNING: You are using pip version 20.3.1; however, version 21.0.1 is
available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip
install --upgrade pip' command.
WARNING: You are using pip version 20.3.1; however, version 21.0.1 is
available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip
install --upgrade pip' command.
```

In [2]:

```
import albumentations as A
from resnest.torch.resnet import ResNet, Bottleneck
import random
from glob import glob
from collections import OrderedDict
import os.path as osp
import os
from pytorch_lightning.loggers import TensorBoardLogger
from pytorch_lightning.callbacks.early_stopping import EarlyStopping
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning import LightningModule
from pytorch_lightning import Trainer
from skimage.transform import resize
from torchvision.models import resnet18, resnet34, resnet50
from resnest.torch import resnest50
from tqdm.auto import tqdm
import colorednoise as cn
import librosa
import torchaudio
import torch.nn.functional as F
from torch.utils.data import WeightedRandomSampler
from torch import nn
from torch.utils.data import Dataset, DataLoader
import torchvision
import torch
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score, confusion_matrix
import matplotlib
matplotlib.use('Agg')
```

```
/opt/conda/lib/python3.7/site-packages/torchaudio/backend/utils.py:54:
UserWarning: "sox" backend is being deprecated. The default backend wi
ll be changed to "sox_io" backend in 0.8.0 and "sox" backend will be r
emoved in 0.9.0. Please migrate to "sox_io" backend. Please refer to h
ttps://github.com/pytorch/audio/issues/903 for the detail.
  '"sox" backend is being deprecated. '
```

In [3]:

```
def seed_everything(seed=42):
    print(f'setting everything to seed {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
    torch.cuda.empty_cache()

seed_everything(42)
```

```
setting everything to seed 42
```

In [4]:

```
# https://www.kaggle.com/c/rfcx-species-audio-detection/discussion/198
# label-level average
# Assume float preds [BxC], labels [BxC] of 0 or 1
def LWLRAP(preds, labels):
    # Ranks of the predictions
    ranked_classes = torch.argsort(preds, dim=-1, descending=True)
    # i, j corresponds to rank of prediction in row i
    class_ranks = torch.zeros_like(ranked_classes).to(preds.device)
    for i in range(ranked_classes.size(0)):
        for j in range(ranked_classes.size(1)):
            class_ranks[i, ranked_classes[i][j]] = j + 1
    # Mask out to only use the ranks of relevant GT labels
    ground_truth_ranks = class_ranks * labels + (1e6) * (1 - labels)
    # All the GT ranks are in front now
    sorted_ground_truth_ranks, _ = torch.sort(
        ground_truth_ranks, dim=-1, descending=False)
    # Number of GT labels per instance
    num_labels = labels.sum(-1)
    pos_matrix = torch.tensor(
        np.array([i+1 for i in range(labels.size(-1))])).unsqueeze(0).
    score_matrix = pos_matrix / sorted_ground_truth_ranks
    score_mask_matrix, _ = torch.sort(labels, dim=-1, descending=True)
    scores = score_matrix * score_mask_matrix
    score = scores.sum() / labels.sum()
    return score.item()
```

In [5]:

```
class Config:
    batch_size = 8
    weight_decay = 1e-8
    lr = 1e-3
    num_workers = 4
    epochs = 6
    num_classes = 24
    sr = 32_000
    duration = 5
    total_duration = 60
    nmels = 128
    EXTRAS_DIR = "../input/rfcxextras"
    ROOT = "../input/rfcx-species-audio-detection"
    TRAIN_AUDIO_ROOT = osp.join(ROOT, "train")
    TEST_AUDIO_ROOT = osp.join(ROOT, "test")
    loss_fn = torch.nn.BCEWithLogitsLoss()
```

Audio Augmentations

In [6]:

```
# Mostly taken from https://www.kaggle.com/hidehisaarai1213/rfcx-audic
class AudioTransform:
    def __init__(self, always_apply=False, p=0.5):
        self.always_apply = always_apply
        self.p = p

    def __call__(self, y: np.ndarray):
        if self.always_apply:
            return self.apply(y)
        else:
            if np.random.rand() < self.p:
                return self.apply(y)
            else:
                return y

    def apply(self, y: np.ndarray):
        raise NotImplementedError


class Compose:
    def __init__(self, transforms: list):
        self.transforms = transforms

    def __call__(self, y: np.ndarray):
        for trns in self.transforms:
            y = trns(y)
        return y


class OneOf:
    def __init__(self, transforms: list):
        self.transforms = transforms

    def __call__(self, y: np.ndarray):
        n_trns = len(self.transforms)
        trns_idx = np.random.choice(n_trns)
        trns = self.transforms[trns_idx]
        return trns(y)


class GaussianNoiseSNR(AudioTransform):
    def __init__(self, always_apply=False, p=0.5, min_snr=5.0, max_snr
        super().__init__(always_apply, p)

        self.min_snr = min_snr
        self.max_snr = max_snr

    def apply(self, y: np.ndarray, **params):
        snr = np.random.uniform(self.min_snr, self.max_snr)
        a_signal = np.sqrt(y ** 2).max()
        a_noise = a_signal / (10 ** (snr / 20))

        white_noise = np.random.randn(len(y))
        a_white = np.sqrt(white_noise ** 2).max()
        augmented = (y + white_noise * 1 / a_white * a_noise).astype(y
        return augmented


class PinkNoiseSNR(AudioTransform):
    def __init__(self, always_apply=False, p=0.5, min_snr=5.0, max_snr
        super().__init__(always_apply, p)
        pink_noise = cn.powerlaw_psd_gaussian(1, len(y))
        a_pink = np.sqrt(pink_noise ** 2).max()
        augmented = (y + pink_noise * 1 / a_pink * a_noise).astype(y.d
        return augmented


class TimeShift(AudioTransform):
    def __init__(self, always_apply=False, p=0.5, max_shift_second=2,
        super().__init__(always_apply, p)

        assert padding_mode in [
            "replace", "zero"], "`padding_mode` must be either 'replac
        self.max_shift_second = max_shift_second
        self.sr = sr
        self.padding_mode = padding_mode

    def apply(self, y: np.ndarray, **params):
        shift = np.random.randint(-self.sr * self.max_shift_second,
                                  self.sr * self.max_shift_second)
        augmented = np.roll(y, shift)
        # if self.padding_mode == "zero":
        #     if shift > 0:
        #         augmented[:shift] = 0
        #     else:
        #         augmented[shift:] = 0
        return augmented


class VolumeControl(AudioTransform):
    def __init__(self, always_apply=False, p=0.5, db_limit=10, mode="u
        super().__init__(always_apply, p)

        assert mode in ["uniform", "fade", "fade", "cosine", "sine"],
            "`mode` must be one of 'uniform', 'fade', 'cosine', 'sine'

        self.db_limit = db_limit
        self.mode = mode

    def apply(self, y: np.ndarray, **params):
        db = np.random.uniform(-self.db_limit, self.db_limit)
        if self.mode == "uniform":
            db_translated = 10 ** (db / 20)
        elif self.mode == "fade":
            lin = np.arange(len(y))[::-1] / (len(y) - 1)
            db_translated = 10 ** (db * lin / 20)
        elif self.mode == "cosine":
            cosine = np.cos(np.arange(len(y)) / len(y) * np.pi * 2)
            db_translated = 10 ** (db * cosine / 20)
        else:
            sine = np.sin(np.arange(len(y)) / len(y) * np.pi * 2)
            db_translated = 10 ** (db * sine / 20)
        augmented = y * db_translated
        return augmented
```

In [7]:

In [8]:

```
    # Normalize to [0, 255]
class RFCDataset:
    def __init__(self, tp, fp=None, config=None,
                 mode='train', inv_counts=None):
        self.tp = tp
        self.fp = pd.read_csv("../input/rfcxextras/cornell-train.csv")
        self.fp = self.fp[self.fp.ebird_code<'c'].reset_index(drop=Tru
        self.fp_root = "../input/birdsong-resampled-train-audio-00/"
        self.inv_counts = inv_counts
        self.config = config
        self.sr = self.config.sr
        self.total_duration = self.config.total_duration
        self.duration = self.config.duration
        self.data_root = self.config.TRAIN_AUDIO_ROOT
        self.nmels = self.config.nmels
        self.fmin, self.fmax = 84, self.sr//2
        self.mode = mode
        self.num_classes = self.config.num_classes
        self.resampler = torchaudio.transforms.Resample(
            orig_freq=48_000, new_freq=self.sr)
        self.mel = torchaudio.transforms.MelSpectrogram(sample_rate=se
                                                        f_min=self.fmi
                                                        n_fft=2048)
        self.transform = Compose([
            OneOf([
                GaussianNoiseSNR(min_snr=10),
                PinkNoiseSNR(min_snr=10)
            ]),
            TimeShift(sr=self.sr),
            VolumeControl(p=0.5)
        ])
        self.img_transform = A.Compose([
            A.OneOf([
                A.Cutout(max_h_size=5, max_w_size=20),
                A.CoarseDropout(max_holes=4),
                A.RandomBrightness(p=0.25),
            ], p=0.5)])
        self.num_splits = self.config.total_duration//self.duration
        assert self.config.total_duration == self.duration * \
            self.num_splits, "not a multiple"

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

    def __getitem__(self, idx):
        labels = np.zeros((self.num_classes,), dtype=np.float32)

        recording_id = self.tp.loc[idx, 'recording_id']
        df = self.tp.loc[self.tp.recording_id == recording_id]
        maybe_labels = df.species_id.unique()
        np.put(labels, maybe_labels, 0.2)

        df = df.sample(weights=df.species_id.apply(
            lambda x: self.inv_counts[x]))
        fn = osp.join(self.data_root, f"{recording_id}.flac")
        df = df.squeeze()
        t0 = max(df['t_min'], 0)
        t1 = max(df['t_max'], 0)
        t0 = np.random.uniform(t0, t1)
        t0 = max(t0, 0)
        t0 = min(t0, self.total_duration-self.duration)
        t1 = t0 + self.duration
        valid_df = self.tp[self.tp.recording_id == recording_id]
        if random.random()<0.5:
            end_idx = int((valid_df.t_max.max() - t0)*self.sr)
            rem_len = max(0, len(y) - end_idx)
            idx = np.random.randint(0, len(self.fp))

            fn = osp.join(self.fp_root, self.fp.ebird_code[idx],self.f
            fn = fn.replace('mp3', 'wav')
            y_other, _ = librosa.load(fn, sr=self.sr,
                                    duration=None, mono=True,
                                    res_type='kaiser_fast')
            aug_len = min(len(y_other), rem_len)
            y[end_idx:end_idx+aug_len] = y_other[:aug_len]

        y = self.resampler(torch.from_numpy(y).float()).numpy()
        # do augmentation
        y = self.transform(y)
        if random.random() < 0.25:
            tempo, beats = librosa.beat.beat_track(y=y, sr=self.sr)
            y = librosa.clicks(frames=beats, sr=self.sr, length=len(y)

        melspec = librosa.feature.melspectrogram(
            y, sr=self.sr, n_mels=self.nmels, fmin=self.fmin, fmax=sel
        )
        melspec = librosa.power_to_db(melspec)
        melspec = mono_to_color(melspec)
        melspec = normalize(melspec, mean=None, std=None)
        melspec = self.img_transform(image=melspec)['image']
        melspec = np.moveaxis(melspec, 2, 0)
        return melspec, labels
```

In [9]:

```
class RFCTestDataset:
    def __init__(self, tp, fp=None, config=None,
                 mode='test'):
        self.tp = tp
        self.fp = fp
        self.config = config
        self.sr = self.config.sr
        self.duration = self.config.duration
        if mode == 'val':
            self.data_root = self.config.TRAIN_AUDIO_ROOT
        else:
            self.data_root = self.config.TEST_AUDIO_ROOT

        self.nmels = self.config.nmels
        self.fmin, self.fmax = 84, self.sr//2
        self.mode = mode
        self.resampler = torchaudio.transforms.Resample(
            orig_freq=48_000, new_freq=self.sr)
        self.num_classes = self.config.num_classes
        self.num_splits = self.config.total_duration//self.duration
        assert self.config.total_duration == self.duration * \
            self.num_splits, "not a multiple"

    def __len__(self):
        return len(self.tp.recording_id.unique())
```

```
            fn = f"{self.config.EXTRAS_DIR}/test_melspec32k_10s/test_m
        try:
            melspec_stacked = np.load(fn)
        except:
            audio_fn = osp.join(self.data_root, f"{recording_id}.flac"
            y, _ = librosa.load(audio_fn, sr=None
```

In [10]:

```
# resnest 50 trained on cornell
# https://www.kaggle.com/theoviel/birds-cp-1
MODEL_CONFIGS = {
    "resnest50_fast_1s1x64d":
    {
        "num_classes": 264,
        "block": Bottleneck,
        "layers": [3, 4, 6, 3],
        "radix": 1,
        "groups": 1,
        "bottleneck_width": 64,
        "deep_stem": True,
        "stem_width": 32,
        "avg_down": True,
        "avd": True,
        "avd_first": True
    }
}


def get_model(pretrained=True, n_class=24):
    # model = torchvision.models.resnext50_32x4d(pretrained=False)
    # model = torchvision.models.resnext101_32x8d(pretrained=False)
    model = ResNet(**MODEL_CONFIGS["resnest50_fast_1s1x64d"])
    n_features = model.fc.in_features
    model.fc = nn.Linear(n_features, 264)
    # model.load_state_dict(torch.load('resnext50_32x4d_extra_2.pt'))
    # model.load_state_dict(torch.load('resnext101_32x8d_wsl_extra_4.p
    fn = '../input/birds-cp-1/resnest50_fast_1s1x64d_conf_1.pt'
    model.load_state_dict(torch.load(fn, map_location='cpu'))
    model.fc = nn.Linear(n_features, n_class)
    return model
```

In [11]:

```
class BaseNet(LightningModule):
    def __init__(self, config, train_recid, val_recid):
        super().__init__()
        self.config = config
        self.batch_size = self.config.batch_size
        self.num_workers = self.config.num_workers
        self.lr = self.config.lr
        self.epochs = self.config.epochs

        self.weight_decay = self.config.weight_decay
        # to improve species 3 recall rate
        pos_weight = torch.ones((24,))
        pos_weight[3] = 4
        self.loss_fn = torch.nn.BCEWithLogitsLoss(pos_weight=pos_weigh
        self.sr = self.config.sr
        self.train_recid = train_recid
        self.val_recid = val_recid

    def train_dataloader(self):
        tp = train_tp[train_tp.recording_id.isin(
            self.train_recid)].reset_index(drop=True)
        self.train_recid = tp.recording_id.unique()
        inv_counts = dict(1/tp.species_id.value_counts())
        weights = tp.species_id.apply(lambda x: inv_counts[x])
        tp_aug = new_labels[new_labels.recording_id.isin(tp.recording_
        tp = pd.concat([tp, tp_aug], ignore_index=True)
        train_dataset = RFCDataset(tp, train_fp,
                                   config=self.config,
                                   mode='train',
                                   inv_counts=inv_counts)
        train_sampler = WeightedRandomSampler(weights, num_samples=len
                                              replacement=True)

        train_loader = DataLoader(train_dataset, batch_size=self.batch
                                  num_workers=self.num_workers,
                                  sampler=train_sampler,
                                  drop_last=True,
                                  pin_memory=True)
        return train_loader

    def val_dataloader(self):
        val_tp = train_tp[train_tp.recording_id.isin(
            self.val_recid)].reset_index(drop=True)
        val_recid = val_tp.recording_id.unique()
        overlap = set(val_recid).intersection(set(self.train_recid))
#         print('overlapped ids', overlap)
        val_tp = val_tp[~val_tp.recording_id.isin(overlap)]
        val_tp_aug = new_labels[new_labels.recording_id.isin(
            val_tp.recording_id)]
        val_tp = pd.concat([val_tp, val_tp_aug], ignore_index=True)
        val_dataset = RFCTestDataset(val_tp, train_fp,
                                     config=self.config,
                                     mode='val')
        val_loader = DataLoader(val_dataset, batch_size=self.batch_siz
                                num_workers=self.num_workers, shuffle=
                                pin_memory=True)
        return val_loader

    def configure_optimizers(self):
        optim = torch.optim.AdamW(self.parameters(), lr=self.config.lr
                                  weight_decay=self.config.weight_deca
        scheduler = {
            'frequency': 1,
            'strict': True,
        }

        self.optimizer = optim
        self.scheduler = scheduler

        return [optim], [scheduler]
```

In [12]:

```
class RFCNet(BaseNet):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        n_class = self.config.num_classes
        self.model = get_model(
            pretrained=True, n_class=n_class)
        self.cnf_matrix = np.zeros((n_class, n_class))

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        x, y = batch
        preds = self(x)
        loss = self.loss_fn(preds, y)
        with torch.no_grad():
            lwlrap = LWLRAP(preds, y)
        metrics = {"train_loss": loss.item(), "train_lwlrap": lwlrap}
        self.log_dict(metrics,
                      on_epoch=True, on_step=True)

        return loss

    @torch.no_grad()
    def validation_step(self, batch, batch_idx):
        x, y = batch
        for i, x_partial in enumerate(torch.split(x, 1, dim=1)):
            x_partial = x_partial.squeeze(1)
            if i == 0:
                preds = self(x_partial)
            else:
                # take max over predictions
                preds = torch.max(preds, self(x_partial))
        val_loss = self.loss_fn(preds, y).item()
        val_lwlrap = LWLRAP(preds, y)
        # loss is tensor. The Checkpoint Callback is monitoring 'check
        metrics = {"val_loss": val_loss, "val_lwlrap": val_lwlrap}
        self.log_dict(metrics, prog_bar=True,
                      on_epoch=True, on_step=True)
```

Average model weights

In [13]:

```
def average_model(paths):
    weights = np.ones((len(paths),))
    weights = weights/weights.sum()
    for i, p in enumerate(paths):
        m = torch.load(p)['state_dict']
        if i == 0:
            averaged_w = OrderedDict()
            for k in m.keys():
                if 'pos' in k: continue
                # remove pl prefix in state dict
                knew = k.replace('model.', '')
                averaged_w[knew] = weights[i]*m[k]
        else:
            for k in m.keys():
                if 'pos' in k: continue
                knew = k.replace('model.', '')
                averaged_w[knew] = averaged_w[knew] + weights[i]*m[k]
    return averaged_w
```

Model training

In [14]:

```python
config = Config()
train_tp = pd.read_csv(osp.join(config.ROOT, 'train_tp.csv'))

fold_df = pd.read_csv(
    osp.join(config.EXTRAS_DIR, 'preprocessed_rainforest_dataset.csv')
fn = "../input/extra-labels-for-rcfx-competition-data/extra_labels_v71
print(fn)
new_labels = pd.read_csv(fn)
new_labels['t_diff'] = new_labels['t_max'] - new_labels['t_min']
idx = np.where(new_labels['t_diff'] < 0)[0]
new_labels = new_labels.drop(idx, axis=0).reset_index(drop=True)
num_folds = len(fold_df.fold.unique())
train_fp = pd.read_csv(osp.join(config.ROOT, 'train_fp.csv'))
for fold in range(num_folds):
    print('\n\nTraining fold', fold)
    print('*' * 40)

    train_recid = fold_df[fold_df.fold != fold].recording_id
    val_recid = fold_df[fold_df.fold == fold].recording_id
    model = RFCNet(config=config, train_recid=train_recid,
                   val_recid=val_recid)
    checkpoint_callback = ModelCheckpoint(
        monitor='val_lwlrap_epoch',
        filename='{epoch:02d}-{val_loss_epoch:.2f}-{val_lwlrap_epoch:.
        mode='max',
        save_top_k=5,
        save_weights_only=True,
    )
    early_stopping = EarlyStopping(monitor='val_lwlrap_epoch', mode='m
                                   verbose=True)
    trainer = Trainer(gpus=1,
                      max_epochs=config.epochs,
                      progress_bar_refresh_rate=1,
                      #   gradient_clip_val=2,
                      accumulate_grad_batches=4,
                      num_sanity_val_steps=0,
                      callbacks=[checkpoint_callback, early_stopping])

    trainer.fit(model)
```

```
../input/extra-labels-for-rcfx-competition-data/extra_labels_v71.csv


Training fold 0
****************************************
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name    | Type              | Params
---------------------------------------------
0 | loss_fn | BCEWithLogitsLoss | 0
1 | model   | ResNet            | 24.2 M
```

Epoch 5: 100%

1008/1008 [32:22<00:00, 1.93s/it, loss=0.138, v_num=0, val_loss_step=0.0164,

val_lwlrap_step=1, val_loss_epoch=0.15, val_lwlrap_epoch=0.947]

Validating: 100% 29/29 [00:08<00:00, 5.35it/s]

```
Validating: 100%                                          29/29 [00:08<00:00, 5.21it/s]

Validating: 100%                                          29/29 [00:08<00:00, 6.12it/s]

Validating: 100%                                          29/29 [00:07<00:00, 5.41it/s]

Validating: 100%                                          29/29 [00:07<00:00, 5.37it/s]

Validating: 100%                                          29/29 [00:07<00:00, 5.41it/s]
```

```
Training fold 1
****************************************
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name    | Type              | Params
----------------------------------------------
0 | loss_fn | BCEWithLogitsLoss | 0
1 | model   | ResNet            | 24.2 M
```

```
Epoch 5: 100%

1009/1009 [33:21<00:00, 1.98s/it, loss=0.131, v_num=1, val_loss_step=0.214,

val_lwlrap_step=0.832, val_loss_epoch=0.141, val_lwlrap_epoch=0.945]

Validating: 100%                                          28/28 [00:07<00:00, 5.43it/s]

Validating: 100%                                          28/28 [00:07<00:00, 5.42it/s]

Validating: 100%                                          28/28 [00:07<00:00, 5.41it/s]

Validating: 100%                                          28/28 [00:07<00:00, 5.41it/s]

Validating: 100%                                          28/28 [00:07<00:00, 5.45it/s]
```

```
Epoch     5: reducing learning rate of group 0 to 5.0000e-04.
```

```
Validating: 100%                                          28/28 [00:07<00:00, 5.43it/s]
```

```
Training fold 2
****************************************
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name    | Type              | Params
----------------------------------------------
0 | loss_fn | BCEWithLogitsLoss | 0
1 | model   | ResNet            | 24.2 M
```

```
Epoch 5: 100%

1013/1013 [28:47<00:00, 1.71s/it, loss=0.140, v_num=2, val_loss_step=0.129,

val_lwlrap_step=0.917, val_loss_epoch=0.147, val_lwlrap_epoch=0.94]
```

```
Validating: 100%                                   29/29 [00:07<00:00, 5.36it/s]

Validating: 100%                                   29/29 [00:07<00:00, 5.27it/s]

Validating: 100%                                   29/29 [00:07<00:00, 5.80it/s]

Validating: 100%                                   29/29 [00:07<00:00, 5.39it/s]

Validating: 100%                                   29/29 [00:07<00:00, 5.43it/s]

Validating: 100%                                   29/29 [00:07<00:00, 5.41it/s]
```

```
Training fold 3
****************************************
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name    | Type              | Params
----------------------------------------------
0 | loss_fn | BCEWithLogitsLoss | 0
1 | model   | ResNet            | 24.2 M
```

```
Epoch 5: 100%

1012/1012 [28:17<00:00, 1.68s/it, loss=0.131, v_num=3, val_loss_step=0.14,

val_lwlrap_step=0.955, val_loss_epoch=0.144, val_lwlrap_epoch=0.94]

Validating: 100%                                   29/29 [00:07<00:00, 5.78it/s]

Validating: 100%                                   29/29 [00:07<00:00, 5.80it/s]

Validating: 100%                                   29/29 [00:08<00:00, 5.78it/s]

Validating: 100%                                   29/29 [00:08<00:00, 5.79it/s]

Validating: 100%                                   29/29 [00:07<00:00, 5.80it/s]

Validating: 100%                                   29/29 [00:08<00:00, 5.80it/s]
```

```
Training fold 4
****************************************
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name    | Type              | Params
----------------------------------------------
0 | loss_fn | BCEWithLogitsLoss | 0
1 | model   | ResNet            | 24.2 M
```

```
Epoch 5: 100%

1031/1031 [08:24<00:00, 2.04it/s, loss=0.124, v_num=4, val_loss_step=0.203,

val_lwlrap_step=0.928, val_loss_epoch=0.12, val_lwlrap_epoch=0.956]
```

```
Validating: 100%                                      29/29 [00:07<00:00, 5.84it/s]

Validating: 100%                                      29/29 [00:08<00:00, 6.01it/s]

Validating: 100%                                      29/29 [00:08<00:00, 6.04it/s]

Validating: 100%                                      29/29 [00:08<00:00, 6.03it/s]
Epoch     4: reducing learning rate of group 0 to 5.0000e-04.
Validating: 100%                                      29/29 [00:08<00:00, 5.95it/s]
```

Model Validation

In [15]:

```
def get_one_hot(targets, nb_classes=24):
    res = np.eye(nb_classes)[np.array(targets).reshape(-1)]
    return res.reshape(list(targets.shape)+[nb_classes])


sub = pd.read_csv(osp.join(config.ROOT, 'sample_submission.csv'))
species_cols = list(sub.columns)
species_cols.remove('recording_id')

cv_preds = pd.DataFrame(columns=species_cols)
cv_preds['recording_id'] = train_tp['recording_id'].drop_duplicates()
cv_preds = cv_preds.set_index('recording_id')

label_df = pd.DataFrame(columns=species_cols)
label_df['recording_id'] = train_tp['recording_id'].drop_duplicates()
label_df = label_df.set_index('recording_id')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = get_model(pretrained=False)
model.to(device)
for fold in range(num_folds):
    paths = glob(f"./lightning_logs/version_{fold}/checkpoints/*.ckpt"
    print(paths)
    averaged_w = average_model(paths)
    model.load_state_dict(averaged_w)
    model.eval()
    train_recid = fold_df[fold_df.fold!=fold].recording_id
    val_recid = fold_df[fold_df.fold==fold].recording_id

    val_tp = train_tp[train_tp.recording_id.isin(val_recid)].reset_ind
    val_recid = val_tp.recording_id.unique()
    overlap = set(val_recid).intersection(set(train_recid))
    val_tp = val_tp[~val_tp.recording_id.isin(overlap)]
    val_tp_aug = new_labels[new_labels.recording_id.isin(val_tp.record
    val_tp = pd.concat([val_tp, val_tp_aug], ignore_index=True)

    dataset = RFCTestDataset(val_tp, config=config, mode='val')
    test_loader = DataLoader(dataset, batch_size=config.batch_size,
                             num_workers=config.num_workers,
                             shuffle=False, drop_last=False)
    tk = test_loader
    with torch.no_grad():
        fold_preds, labels = [], []
        for i, (im, l) in enumerate(tk):
            # continue
            im = im.to(device)
            for j, x_partial in enumerate(torch.split(im, 1, dim=1)):
                x_partial = x_partial.squeeze(1)
                if j == 0:
                    preds = model(x_partial)
                else:
                    preds = torch.max(preds, model(x_partial))


            o = preds.sigmoid().cpu().numpy()
            # o = preds.cpu().numpy()
            fold_preds.extend(o)
            labels.extend(l.cpu().numpy())
        # continue
        p = torch.from_numpy(np.array(fold_preds))
        t = torch.from_numpy(np.array(labels))
```

```
recid = train_tp['recording_id'].values
cv_preds = cv_preds.loc[recid].values.astype(np.float32)
cv_preds = torch.from_numpy(cv_preds)

labels = label_df.loc[recid].values.astype(np.float32)
labels = torch.from_numpy(labels)

print(f"lwlrap: {LWLRAP(cv_preds, labels):.6}")
```

```
['./lightning_logs/version_0/checkpoints/epoch=02-val_loss_epoch=0.12-
val_lwlrap_epoch=0.96.ckpt', './lightning_logs/version_0/checkpoints/e
poch=04-val_loss_epoch=0.14-val_lwlrap_epoch=0.95.ckpt', './lightning_
logs/version_0/checkpoints/epoch=03-val_loss_epoch=0.13-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_0/checkpoints/epoch=00-val_lo
ss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt', './lightning_logs/version_0
/checkpoints/epoch=05-val_loss_epoch=0.15-val_lwlrap_epoch=0.95.ckpt']
lwlrap: 0.9604
['./lightning_logs/version_1/checkpoints/epoch=05-val_loss_epoch=0.14-
val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_1/checkpoints/e
poch=02-val_loss_epoch=0.14-val_lwlrap_epoch=0.94.ckpt', './lightning_
logs/version_1/checkpoints/epoch=04-val_loss_epoch=0.14-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_1/checkpoints/epoch=01-val_lo
ss_epoch=0.13-val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_1
/checkpoints/epoch=03-val_loss_epoch=0.15-val_lwlrap_epoch=0.94.ckpt']
lwlrap: 0.95972
['./lightning_logs/version_2/checkpoints/epoch=02-val_loss_epoch=0.15-
val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_2/checkpoints/e
poch=01-val_loss_epoch=0.14-val_lwlrap_epoch=0.95.ckpt', './lightning_
logs/version_2/checkpoints/epoch=00-val_loss_epoch=0.13-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_2/checkpoints/epoch=05-val_lo
ss_epoch=0.15-val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_2
/checkpoints/epoch=03-val_loss_epoch=0.15-val_lwlrap_epoch=0.94.ckpt']
lwlrap: 0.964733
['./lightning_logs/version_3/checkpoints/epoch=05-val_loss_epoch=0.14-
val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_3/checkpoints/e
poch=03-val_loss_epoch=0.14-val_lwlrap_epoch=0.94.ckpt', './lightning_
logs/version_3/checkpoints/epoch=02-val_loss_epoch=0.13-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_3/checkpoints/epoch=00-val_lo
ss_epoch=0.14-val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_3
/checkpoints/epoch=01-val_loss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt']
lwlrap: 0.957236
['./lightning_logs/version_4/checkpoints/epoch=02-val_loss_epoch=0.12-
val_lwlrap_epoch=0.96.ckpt', './lightning_logs/version_4/checkpoints/e
poch=03-val_loss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt', './lightning_
logs/version_4/checkpoints/epoch=05-val_loss_epoch=0.12-val_lwlrap_epo
ch=0.96.ckpt', './lightning_logs/version_4/checkpoints/epoch=04-val_lo
ss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt', './lightning_logs/version_4
/checkpoints/epoch=01-val_loss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt']
lwlrap: 0.96354
lwlrap: 0.960364
```

Test predictions

In [16]:

```
sub = pd.read_csv(osp.join(config.ROOT, 'sample_submission.csv'))
species_cols = list(sub.columns)
species_cols.remove('recording_id')
# initialize to zero.
sub.loc[:, species_cols] = 0

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = get_model(pretrained=False)
model.to(device)
for fold in range(num_folds):
    paths = glob(f"./lightning_logs/version_{fold}/checkpoints/*.ckpt"
    print(paths)
    averaged_w = average_model(paths)
    model.load_state_dict(averaged_w)
    model.eval()
    dataset = RFCTestDataset(sub, config=config, mode='test')
    test_loader = DataLoader(dataset, batch_size=config.batch_size,
                             num_workers=4,
                             shuffle=False, drop_last=False)
    tk = tqdm(test_loader, total=len(test_loader))
    sub_index = 0
    with torch.no_grad():
        for i, im in enumerate(tk):
            im = im.to(device)
            for i, x_partial in enumerate(torch.split(im, 1, dim=1)):
                x_partial = x_partial.squeeze(1)
                if i == 0:
                    preds = model(x_partial)
                else:
                    # take max over predictions
                    preds = torch.max(preds, model(x_partial))

            o = preds.sigmoid().cpu().numpy()
            # o = preds.cpu().numpy()
            for val in o:
                sub.loc[sub_index, species_cols] += val
                sub_index += 1

# # take average of predictions
sub.loc[:, species_cols] /= num_folds
sub.to_csv('submission.csv', index=False)
print(sub.head())
print(sub.max(1).head())
```

```
['./lightning_logs/version_0/checkpoints/epoch=02-val_loss_epoch=0.12-
val_lwlrap_epoch=0.96.ckpt', './lightning_logs/version_0/checkpoints/e
poch=04-val_loss_epoch=0.14-val_lwlrap_epoch=0.95.ckpt', './lightning_
logs/version_0/checkpoints/epoch=03-val_loss_epoch=0.13-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_0/checkpoints/epoch=00-val_lo
ss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt', './lightning_logs/version_0
/checkpoints/epoch=05-val_loss_epoch=0.15-val_lwlrap_epoch=0.95.ckpt']
```

```
100%                                   249/249 [02:39<00:00, 1.56it/s]
```

```
['./lightning_logs/version_1/checkpoints/epoch=05-val_loss_epoch=0.14-
val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_1/checkpoints/e
poch=02-val_loss_epoch=0.14-val_lwlrap_epoch=0.94.ckpt', './lightning_
logs/version_1/checkpoints/epoch=04-val_loss_epoch=0.14-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_1/checkpoints/epoch=01-val_lo
ss_epoch=0.13-val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_1
/checkpoints/epoch=03-val_loss_epoch=0.15-val_lwlrap_epoch=0.94.ckpt']
```

```
100%                                   249/249 [01:18<00:00, 3.17it/s]
```

```
['./lightning_logs/version_2/checkpoints/epoch=02-val_loss_epoch=0.15-
val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_2/checkpoints/e
poch=01-val_loss_epoch=0.14-val_lwlrap_epoch=0.95.ckpt', './lightning_
logs/version_2/checkpoints/epoch=00-val_loss_epoch=0.13-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_2/checkpoints/epoch=05-val_lo
ss_epoch=0.15-val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_2
/checkpoints/epoch=03-val_loss_epoch=0.15-val_lwlrap_epoch=0.94.ckpt']
```

```
100%                                   249/249 [01:16<00:00, 3.27it/s]
```

```
['./lightning_logs/version_3/checkpoints/epoch=05-val_loss_epoch=0.14-
val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_3/checkpoints/e
poch=03-val_loss_epoch=0.14-val_lwlrap_epoch=0.94.ckpt', './lightning_
logs/version_3/checkpoints/epoch=02-val_loss_epoch=0.13-val_lwlrap_epo
ch=0.95.ckpt', './lightning_logs/version_3/checkpoints/epoch=00-val_lo
ss_epoch=0.14-val_lwlrap_epoch=0.94.ckpt', './lightning_logs/version_3
/checkpoints/epoch=01-val_loss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt']
```

```
100%                                   249/249 [01:16<00:00, 3.23it/s]
```

```
['./lightning_logs/version_4/checkpoints/epoch=02-val_loss_epoch=0.12-
val_lwlrap_epoch=0.96.ckpt', './lightning_logs/version_4/checkpoints/e
poch=03-val_loss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt', './lightning_
logs/version_4/checkpoints/epoch=05-val_loss_epoch=0.12-val_lwlrap_epo
ch=0.96.ckpt', './lightning_logs/version_4/checkpoints/epoch=04-val_lo
ss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt', './lightning_logs/version_4
/checkpoints/epoch=01-val_loss_epoch=0.13-val_lwlrap_epoch=0.95.ckpt']
```

```
100%                                   249/249 [01:16<00:00, 3.26it/s]
```

```
  recording_id        s0        s1        s2        s3        s4      
s5  \
0    000316da7  0.279858  0.002074  0.005444  0.999666  0.002251  0.03
1033
1    003bc2cb2  0.000077  0.019530  0.000070  0.998492  0.000225  0.00
0478
2    0061c037e  0.002871  0.010663  0.001739  0.994012  0.001690  0.04
6972
3    010eb14d3  0.999818  0.000130  0.005920  0.999975  0.006016  0.00
0148
4    011318064  0.004463  0.031771  0.001013  0.998716  0.012782  0.01
4602

         s6        s7        s8  ...       s14       s15       s16      
s17  \
0  0.002359  0.023500  0.012245  ...  0.048399  0.045740  0.001581  0.
001339
1  0.000914  0.001015  0.000159  ...  0.000027  0.003628  0.999838  0.
003442
2  0.042452  0.962784  0.001351  ...  0.000639  0.824202  0.001664  0.
021069
3  0.000018  0.000098  0.999874  ...  0.000012  0.002124  0.000381  0.
000006
4  0.003402  0.008574  0.002033  ...  0.999999  0.999323  0.000521  0.
002278

        s18       s19       s20       s21       s22       s23
0  0.999006  0.007835  0.002340  0.001865  0.001505  0.010295
1  0.003496  0.003759  0.001517  0.000482  0.000146  0.026102
2  0.003494  0.057197  0.394826  0.002553  0.004537  0.107204
3  0.999934  0.001242  0.000056  0.000162  0.000080  0.000083
4  0.982492  0.000661  0.000577  0.002764  0.005496  0.003077

[5 rows x 25 columns]
```

```
0    0.999666
1    0.999838
2    0.994012
3    0.999975
4    0.999999
```

In [17]:

```
sub.iloc[:, 1:].describe()
```

Out[17]:

	s0	s1	s2	s3	s4	s5
count	1992.000000	1992.000000	1.992000e+03	1992.000000	1992.000000	1992.000000
mean	0.144494	0.249642	1.130642e-01	0.971583	0.069028	0.106453
std	0.317923	0.380492	2.681151e-01	0.099464	0.188514	0.238744
min	0.000003	0.000053	7.000626e-07	0.112037	0.000008	0.000031
25%	0.000408	0.003491	8.843963e-04	0.996040	0.000870	0.002063
50%	0.002369	0.022956	4.667131e-03	0.999444	0.004374	0.010422
75%	0.039214	0.365116	3.497232e-02	0.999857	0.026496	0.059821
max	0.999999	0.999999	9.999998e-01	0.999999	0.999982	0.999986

8 rows × 24 columns