This notebook shows the training of RFCX data on Tensorflow TPU

The dataset used in this notebook is 10 fold Groupkfold tp only tfrecords that i have created here and the simple script for the notebook is this.

Training description :

- training with 10 sec clip around true positives
- taking full spectrogram size
- random augmentation and gaussian noise
- label smoothing
- stepwise cosine decay with warm restarts and early stopping
- for inference 10sec clip is used and then aggregrating and taking max of the audio wav prediction

Since this notebook uses tpu accelerator having 128 gb (16 gb each replica) so for efficient use i have done following optimization :

- increased the spectrogram size
- caching validation and test set as both are small in number for faster computation
- wrapped all user defined function with map that allow parallel computation
- reduced the python overhead
- tensorflow 2.3 and above has argument execution per step in model.compile function that significantly improves performance by running multiple steps within tpu worker. but since kaggle has not updated tf version we cannot take advantage of that but one can try it on google colab
- above step can also be done by using custom training loop

In [1]:

```
! pip install -q efficientnet
```

```
WARNING: You are using pip version 20.1.1; however, version 20.3.3 is avail
able.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip inst
all --upgrade pip' command.
```

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

In [2]:

```
import math, os, re, warnings, random
import tensorflow as tf
import numpy as np
import pandas as pd
import librosa
from kaggle_datasets import KaggleDatasets
import matplotlib.pyplot as plt
from IPython.display import Audio
from tensorflow.keras import Model, layers
from sklearn.model_selection import KFold
import tensorflow.keras.backend as K
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, Lea
from tensorflow.keras.layers import GlobalAveragePooling2D, Input, Dense, I
from tensorflow.keras.applications import ResNet50
import efficientnet.keras as efn
import seaborn as sns
```

TPU Detection And Initialization

In [3]:

```
# TPU or GPU detection
# Detect hardware, return appropriate distribution strategy
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print(f'Running on TPU {tpu.master()}')
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    strategy = tf.distribute.get_strategy()

AUTO = tf.data.experimental.AUTOTUNE
REPLICAS = strategy.num_replicas_in_sync
print(f'REPLICAS: {REPLICAS}')
```

```
Running on TPU grpc://10.0.0.2:8470
REPLICAS: 8
```

In [4]:

```
def seed_everything(seed=0):
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'

seed = 42
seed_everything(seed)
warnings.filterwarnings('ignore')
```

In [5]:

```
def count_data_items(filenames):
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1)) for file
    return np.sum(n)

# train_files

TRAIN_DATA_DIR = 'rfcx-audio-detection'
TRAIN_GCS_PATH = KaggleDatasets().get_gcs_path(TRAIN_DATA_DIR)
FILENAMES = tf.io.gfile.glob(TRAIN_GCS_PATH + '/tp*.tfrec')


#test_files
TEST_DATA_DIR = 'rfcx-species-audio-detection'
TEST_GCS_PATH =  KaggleDatasets().get_gcs_path(TEST_DATA_DIR)
TEST_FILES = tf.io.gfile.glob(TEST_GCS_PATH + '/tfrecords/test/*.tfrec')

no_of_training_samples = count_data_items(FILENAMES)

print('num_training_samples are', no_of_training_samples)
```

```
num_training_samples are 1216
```

In [6]:

```
CUT = 10
TIME = 10
EPOCHS = 25
GLOBAL_BATCH_SIZE = 4 * REPLICAS
LEARNING_RATE = 0.0015
WARMUP_LEARNING_RATE = 1e-5
WARMUP_EPOCHS = int(EPOCHS*0.1)
PATIENCE = 8
STEPS_PER_EPOCH = 64
N_FOLDS = 5
NUM_TRAINING_SAMPLES = no_of_training_samples


class params:
    sample_rate = 48000
    stft_window_seconds: float = 0.025
    stft_hop_seconds: float = 0.005
    frame_length: int =  1200
    mel_bands: int = 512
    mel_min_hz: float = 50.0
    mel_max_hz: float = 24000.0
    log_offset: float = 0.001
    patch_window_seconds: float = 0.96
    patch_hop_seconds: float = 0.48


    patch_frames =  int(round(patch_window_seconds / stft_hop_seconds))


    patch_bands = mel_bands
    height = mel_bands
    width = 2000
    num_classes: int = 24
    dropout = 0.35
    classifier_activation: str = 'sigmoid'
```

In [7]:

```
feature_description = {
    'wav': tf.io.FixedLenFeature([], tf.string),
    'recording_id': tf.io.FixedLenFeature([], tf.string ),
    'target' : tf.io.FixedLenFeature([], tf.float32),
    'song_id': tf.io.FixedLenFeature([], tf.float32),
     'tmin' : tf.io.FixedLenFeature([], tf.float32),
     'fmin' : tf.io.FixedLenFeature([], tf.float32),
     'tmax' : tf.io.FixedLenFeature([], tf.float32),
     'fmax' : tf.io.FixedLenFeature([], tf.float32),
}
feature_dtype = {
    'wav': tf.float32,
    'recording_id': tf.string,
    'target': tf.float32,
    'song_id': tf.float32,
    't_min': tf.float32,
    'f_min': tf.float32,
    't_max': tf.float32,
    'f_max':tf.float32,
}
```

In [8]:

```
def waveform_to_log_mel_spectrogram(waveform,target_or_rec_id):
    """Compute log mel spectrogram patches of a 1-D waveform."""
    # waveform has shape [<# samples>]

    # Convert waveform into spectrogram using a Short-Time Fourier Transfo
    # Note that tf.signal.stft() uses a periodic Hann window by default.

    window_length_samples = int(
      round(params.sample_rate * params.stft_window_seconds))
    hop_length_samples = int(
      round(params.sample_rate * params.stft_hop_seconds))
    fft_length = 2 ** int(np.ceil(np.log(window_length_samples) / np.log(2
#     print(fft_length, window_length_samples, hop_length_samples)
    num_spectrogram_bins = fft_length // 2 + 1
    magnitude_spectrogram = tf.abs(tf.signal.stft(
      signals=waveform,
      frame_length=params.frame_length,
      frame_step=hop_length_samples,
      fft_length= fft_length))
    # magnitude_spectrogram has shape [<# STFT frames>, num_spectrogram_bi

    # Convert spectrogram into log mel spectrogram.
    linear_to_mel_weight_matrix = tf.signal.linear_to_mel_weight_matrix(
        num_mel_bins=params.mel_bands,
        num_spectrogram_bins=num_spectrogram_bins,
        sample_rate=params.sample_rate,
        lower_edge_hertz=params.mel_min_hz,
        upper_edge_hertz=params.mel_max_hz)
    mel_spectrogram = tf.matmul(
      magnitude_spectrogram, linear_to_mel_weight_matrix)
    log_mel = tf.math.log(mel_spectrogram + params.log_offset)
#     log_mel_spectrogram has shape [<# STFT frames>, params.mel_bands]
    log_mel = tf.transpose(log_mel)
    log_mel_spectrogram = tf.reshape(log_mel , [tf.shape(log_mel)[0] ,tf.sl
    # Frame spectrogram (shape [<# STFT frames>, params.mel_bands]) into pa
    # (the input examples). Only complete frames are emitted, so if there
    # less than params.patch_window_seconds of waveform then nothing is em
    # (to avoid this, zero-pad before processing).
    spectrogram_hop_length_samples = int(
      round(params.sample_rate * params.stft_hop_seconds))
    spectrogram_sample_rate = params.sample_rate / spectrogram_hop_length_s
    patch_window_length_samples = int(
      round(spectrogram_sample_rate * params.patch_window_seconds))
    patch_hop_length_samples = int(
      round(spectrogram_sample_rate * params.patch_hop_seconds))
    features = tf.signal.frame(
        signal=log_mel_spectrogram,
        frame_length=patch_window_length_samples,
        frame_step=patch_hop_length_samples,
        axis=0)
    # features has shape [<# patches>, <# STFT frames in an patch>, params

    return log_mel_spectrogram, target_or_rec_id
```

Data augmentation

In [9]:

```
def frequency_masking(mel_spectrogram):

    frequency_masking_para = 80,
    frequency_mask_num = 2

    fbank_size = tf.shape(mel_spectrogram)
#     print(fbank_size)
    n, v = fbank_size[0], fbank_size[1]

    for i in range(frequency_mask_num):
        f = tf.random.uniform([], minval=0, maxval= tf.squeeze(frequency_ma
        v = tf.cast(v, dtype=tf.int32)
        f0 = tf.random.uniform([], minval=0, maxval= tf.squeeze(v-f), dtype

        # warped_mel_spectrogram[f0:f0 + f, :] = 0
        mask = tf.concat((tf.ones(shape=(n, v - f0 - f,1)),
                          tf.zeros(shape=(n, f,1)),
                          tf.ones(shape=(n, f0,1)),
                          ),1)
        mel_spectrogram = mel_spectrogram * mask
    return tf.cast(mel_spectrogram, dtype=tf.float32)


def time_masking(mel_spectrogram):
    time_masking_para = 40,
    time_mask_num = 1

    fbank_size = tf.shape(mel_spectrogram)
    n, v = fbank_size[0], fbank_size[1]


    for i in range(time_mask_num):
        t = tf.random.uniform([], minval=0, maxval=tf.squeeze(time_masking_
        t0 = tf.random.uniform([], minval=0, maxval= n-t, dtype=tf.int32)

        # mel_spectrogram[:, t0:t0 + t] = 0
        mask = tf.concat((tf.ones(shape=(n-t0-t, v,1)),
                          tf.zeros(shape=(t, v,1)),
                          tf.ones(shape=(t0, v,1)),
                          ), 0)

        mel_spectrogram = mel_spectrogram * mask
    return tf.cast(mel_spectrogram, dtype=tf.float32)


def random_brightness(image):
    return tf.image.random_brightness(image, 0.2)

def random_gamma(image):
    return tf.image.random_contrast(image, lower = 0.1, upper = 0.3)

def random_flip_right(image):
    return tf.image.random_flip_left_right(image)

def random_flip_up_down(image):
    return tf.image.random_flip_left_right(image)

available_ops = [
          frequency_masking ,
          time_masking,
          random_brightness,
```

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

```
        op_to_select = tf.random.uniform([], maxval=len(available_ops), dty
        for (i, op_name) in enumerate(available_ops):
            image = tf.cond(
            tf.equal(i, op_to_select),
            lambda selected_func=op_name,: selected_func(
                image),
            lambda: image)
    return image, target
```

Training Data Pipeline

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

In [10]:

```
def preprocess(image, target_or_rec_id):

    image = tf.image.grayscale_to_rgb(image)
    image = tf.image.resize(image, [params.height,params.width])
    image = tf.image.per_image_standardization(image)
    return image , target_or_rec_id


def read_labeled_tfrecord(example_proto):
    sample = tf.io.parse_single_example(example_proto, feature_description
    wav, _ = tf.audio.decode_wav(sample['wav'], desired_channels=1) # mono
    target = tf.cast(sample['target'],tf.float32)
    target = tf.squeeze(tf.one_hot([target,], depth = params.num_classes),

    tmin = tf.cast(sample['tmin'], tf.float32)
    fmin = tf.cast(sample['fmin'], tf.float32)
    tmax = tf.cast(sample['tmax'], tf.float32)
    fmax = tf.cast(sample['fmax'], tf.float32)

    tmax_s = tmax * tf.cast(params.sample_rate, tf.float32)
    tmin_s = tmin * tf.cast(params.sample_rate, tf.float32)
    cut_s = tf.cast(CUT * params.sample_rate, tf.float32)
    all_s = tf.cast(60 * params.sample_rate, tf.float32)
    tsize_s = tmax_s - tmin_s
    cut_min = tf.cast(
    tf.maximum(0.0,
        tf.minimum(tmin_s - (cut_s - tsize_s) / 2,
                   tf.minimum(tmax_s + (cut_s - tsize_s) / 2, all_s) - cut_
    ), tf.int32
      )
    cut_max = cut_min + CUT * params.sample_rate
    wav = tf.squeeze(wav[cut_min : cut_max] )

    return wav, target

def read_unlabeled_tfrecord(example):
    feature_description = {
    'recording_id': tf.io.FixedLenFeature([], tf.string),
    'audio_wav': tf.io.FixedLenFeature([], tf.string),
    }
    sample = tf.io.parse_single_example(example, feature_description)
    wav, _ = tf.audio.decode_wav(sample['audio_wav'], desired_channels=1)
    recording_id = tf.reshape(tf.cast(sample['recording_id'] , tf.string),
#     wav = tf.squeeze(wav)

    def _cut_audio(i):
        _sample = {
            'audio_wav': tf.reshape(wav[i*params.sample_rate*TIME:(i+1)*pa
            'recording_id': sample['recording_id']
        }
        return _sample

    return tf.map_fn(_cut_audio, tf.range(60//TIME), dtype={
        'audio_wav': tf.float32,
        'recording_id': tf.string
    })
```

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

In [11]:
```
def load_dataset(filenames, labeled = True, ordered = False , training = Tr
    # Read from TFRecords. For optimal performance, reading from multiple 
    # Diregarding data order. Order does not matter since we will be shuff

    ignore_order = tf.data.Options()
    if not ordered:
        # disable order, increase speed
        ignore_order.experimental_deterministic = False

    # automatically interleaves reads from multiple files
    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads = AUTO
    # use data as soon as it streams in, rather than in its original order
    dataset = dataset.map(read_labeled_tfrecord , num_parallel_calls = AUT(
    dataset = dataset.map(waveform_to_log_mel_spectrogram , num_parallel_ca
    if training:
        dataset = dataset.map(apply_augmentation, num_parallel_calls = AUT(
    dataset = dataset.map(preprocess, num_parallel_calls = AUTO)
    return dataset
```

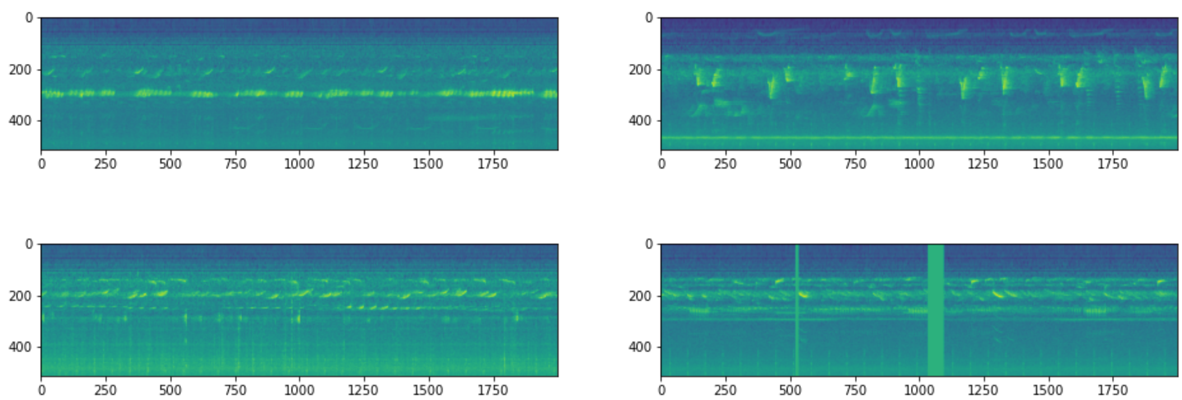
In [12]:
```
def get_dataset(filenames, training = True):
    if training:
        dataset = load_dataset(filenames , training = True)
        dataset = dataset.shuffle(256).repeat()
        dataset = dataset.batch(GLOBAL_BATCH_SIZE, drop_remainder = True)
    else:
        dataset = load_dataset(filenames , training = False)
        dataset = dataset.batch(GLOBAL_BATCH_SIZE).cache()

    dataset = dataset.prefetch(AUTO)
    return dataset
```

In [13]:
```
# mel spectrogram visualization

train_dataset = get_dataset(FILENAMES, training = True)

plt.figure(figsize=(16,6))
for i, (wav, target) in enumerate(train_dataset.unbatch().take(4)):
    plt.subplot(2,2,i+1)
    plt.imshow(wav[:, :, 0])
plt.show()
```



Competition Metric

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

In [14]:

```
# from https://www.kaggle.com/carlthome/l-lrap-metric-for-tf-keras

def _one_sample_positive_class_precisions(example):
    y_true, y_pred = example
    y_true = tf.reshape(y_true, tf.shape(y_pred))
    retrieved_classes = tf.argsort(y_pred, direction='DESCENDING')
#     shape = tf.shape(retrieved_classes)
    class_rankings = tf.argsort(retrieved_classes)
    retrieved_class_true = tf.gather(y_true, retrieved_classes)
    retrieved_cumulative_hits = tf.math.cumsum(tf.cast(retrieved_class_tru

    idx = tf.where(y_true)[:, 0]
    i = tf.boolean_mask(class_rankings, y_true)
    r = tf.gather(retrieved_cumulative_hits, i)
    c = 1 + tf.cast(i, tf.float32)
    precisions = r / c

    dense = tf.scatter_nd(idx[:, None], precisions, [y_pred.shape[0]])
    return dense

# @tf.function
class LWLRAP(tf.keras.metrics.Metric):
    def __init__(self, num_classes, name='lwlrap'):
        super().__init__(name=name)

        self._precisions = self.add_weight(
            name='per_class_cumulative_precision',
            shape=[num_classes],
            initializer='zeros',
        )

        self._counts = self.add_weight(
            name='per_class_cumulative_count',
            shape=[num_classes],
            initializer='zeros',
        )

    def update_state(self, y_true, y_pred, sample_weight=None):
        precisions = tf.map_fn(
            fn=_one_sample_positive_class_precisions,
            elems=(y_true, y_pred),
            dtype=(tf.float32),
        )

        increments = tf.cast(precisions > 0, tf.float32)
        total_increments = tf.reduce_sum(increments, axis=0)
        total_precisions = tf.reduce_sum(precisions, axis=0)

        self._precisions.assign_add(total_precisions)
        self._counts.assign_add(total_increments)

    def result(self):
        per_class_lwlrap = self._precisions / tf.maximum(self._counts, 1.0
        per_class_weight = self._counts / tf.reduce_sum(self._counts)
        overall_lwlrap = tf.reduce_sum(per_class_lwlrap * per_class_weight
        return overall_lwlrap

    def reset_states(self):
        self._precisions.assign(self._precisions * 0)
        self._counts.assign(self._counts * 0)
```

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

Stepwise Cosine Decay Callback

In [15]:

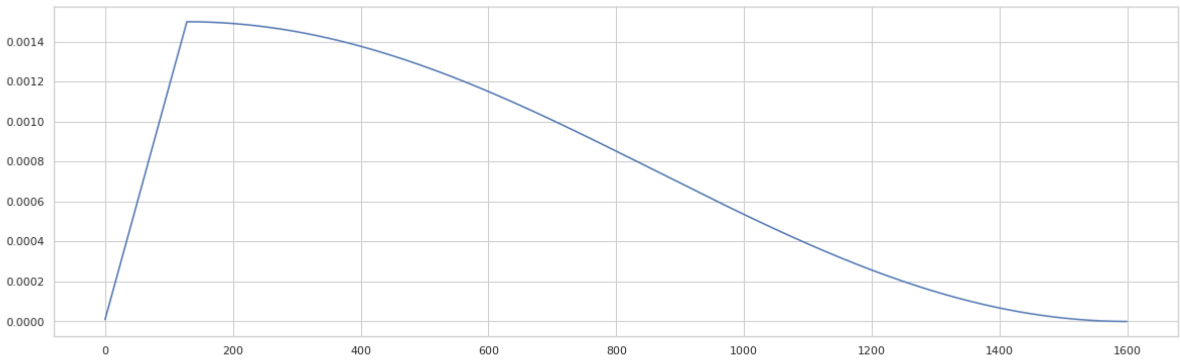
```
def cosine_decay_with_warmup(global_step,
                             learning_rate_base,
                             total_steps,
                             warmup_learning_rate=0.0,
                             warmup_steps= 0,
                             hold_base_rate_steps=0):

    if total_steps < warmup_steps:
        raise ValueError('total_steps must be larger or equal to '
                         'warmup_steps.')
    learning_rate = 0.5 * learning_rate_base * (1 + tf.cos(
        np.pi *
        (tf.cast(global_step, tf.float32) - warmup_steps - hold_base_rate_s
        ) / float(total_steps - warmup_steps - hold_base_rate_steps)))
    if hold_base_rate_steps > 0:
        learning_rate = tf.where(
          global_step > warmup_steps + hold_base_rate_steps,
          learning_rate, learning_rate_base)
    if warmup_steps > 0:
        if learning_rate_base < warmup_learning_rate:
            raise ValueError('learning_rate_base must be larger or equal to
                             'warmup_learning_rate.')
        slope = (learning_rate_base - warmup_learning_rate) / warmup_steps
        warmup_rate = slope * tf.cast(global_step,
                                      tf.float32) + warmup_learning_rate
        learning_rate = tf.where(global_step < warmup_steps, warmup_rate,
                                 learning_rate)
    return tf.where(global_step > total_steps, 0.0, learning_rate,
                    name='learning_rate')


#dummy example
rng = [i for i in range(int(EPOCHS * STEPS_PER_EPOCH))]
WARMUP_STEPS =  int(WARMUP_EPOCHS * STEPS_PER_EPOCH)
y = [cosine_decay_with_warmup(x , LEARNING_RATE, len(rng), 1e-5, WARMUP_STI

sns.set(style='whitegrid')
fig, ax = plt.subplots(figsize=(20, 6))
plt.plot(rng, y)
```

Out[15]: [<matplotlib.lines.Line2D at 0x7ff0fc319550>]



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

In [16]:
```
# to apply learning rate schedule stepwise we need to subclass keras callb
# if we would have applied lr schedule epoch wise then it is not needed we

class WarmUpCosineDecayScheduler(tf.keras.callbacks.Callback):

    def __init__(self,
                 learning_rate_base,
                 total_steps,
                 global_step_init=0,
                 warmup_learning_rate=0.0,
                 warmup_steps=0,
                 hold_base_rate_steps=0,
                 verbose=0):

        super(WarmUpCosineDecayScheduler, self).__init__()
        self.learning_rate_base = learning_rate_base
        self.total_steps = total_steps
        self.global_step = global_step_init
        self.warmup_learning_rate = warmup_learning_rate
        self.warmup_steps = warmup_steps
        self.hold_base_rate_steps = hold_base_rate_steps
        self.verbose = verbose
        self.learning_rates = []

    def on_batch_end(self, batch, logs=None):
        self.global_step = self.global_step + 1
        lr = K.get_value(self.model.optimizer.lr)
        self.learning_rates.append(lr)

    def on_batch_begin(self, batch, logs=None):
        lr = cosine_decay_with_warmup(global_step=self.global_step,
                                      learning_rate_base=self.learning_rate
                                      total_steps=self.total_steps,
                                      warmup_learning_rate=self.warmup_lea
                                      warmup_steps=self.warmup_steps,
                                      hold_base_rate_steps=self.hold_base_
        K.set_value(self.model.optimizer.lr, lr)
        if self.verbose > 0:
            print('\nBatch %05d: setting learning '
                  'rate to %s.' % (self.global_step + 1, lr.numpy()))


total_steps = int(EPOCHS * STEPS_PER_EPOCH)
# Compute the number of warmup batches or steps.
warmup_steps = int(WARMUP_EPOCHS * STEPS_PER_EPOCH)
warmup_learning_rate = WARMUP_LEARNING_RATE
```

Model Definition

In [17]:

```
def RFCX_MODEL():
    waveform = Input(shape=(None,None,3), dtype=tf.float32)
    noisy_waveform = GaussianNoise(0.2)(waveform)
    model = efn.EfficientNetB2(include_top=False, weights='imagenet',)
    model_output = model(noisy_waveform)
    model_output = GlobalAveragePooling2D()(model_output)
    dense = Dropout(params.dropout)(model_output)
    predictions = Dense(params.num_classes, activation = params.classifier_
    model = Model(
      name='Efficientnet', inputs=waveform,
      outputs=[predictions])
    return model
```

In [18]:

```
def get_model():
    with strategy.scope():
        model = RFCX_MODEL()
        model.summary()
        model.compile(optimizer = 'adam',
                        loss = tf.keras.losses.BinaryCrossentropy(
                        metrics = [LWLRAP(num_classes = params.num_
                        ])
    return model
```

Training And Validation Loop

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

In [19]:

```
skf = KFold(n_splits=N_FOLDS, shuffle=True, random_state=seed)
oof_pred = []; oof_labels = []; history_list = []

for fold,(idxT, idxV) in enumerate(skf.split(np.arange(10))):
    if tpu: tf.tpu.experimental.initialize_tpu_system(tpu)
    print(f'\nFOLD: {fold+1}')
    print(f'TRAIN: {idxT} VALID: {idxV}')

    # Create train and validation sets
    TRAIN_FILENAMES = [FILENAMES[x] for x in idxT]
    VALID_FILENAMES = [FILENAMES[x] for x in idxV]
    np.random.shuffle(TRAIN_FILENAMES)

    train_dataset =  get_dataset(TRAIN_FILENAMES, training=True,)
    validation_data= get_dataset(VALID_FILENAMES, training=False)

    model = get_model()

    model_path = f'RFCX_model_fold {fold}.h5'
    early_stopping = EarlyStopping(monitor = 'val_lwlrap', mode = 'max',
                                   patience = PATIENCE, restore_best_weights=True, verl

    # Create the Learning rate scheduler.
    cosine_warm_up_lr = WarmUpCosineDecayScheduler(learning_rate_base= LEAI
                                            total_steps= total_steps,
                                            warmup_learning_rate= warmup_learning_
                                            warmup_steps= warmup_steps,
                                            hold_base_rate_steps=0)

    ## TRAIN
    history = model.fit(train_dataset,
                        steps_per_epoch=STEPS_PER_EPOCH,
                        callbacks=[early_stopping, cosine_warm_up_lr],
                        epochs=EPOCHS,
                        validation_data = validation_data,
                        verbose = 2).history

    history_list.append(history)
    # Save last model weights
    model.save_weights(model_path)

# OOF predictions
    ds_valid = get_dataset(VALID_FILENAMES, training = False)
    oof_labels.append([target.numpy() for frame, target in iter(ds_valid.ur
    x_oof = ds_valid.map(lambda frames, target: frames)
    oof_pred.append(np.argmax(model.predict(x_oof), axis=-1))

    ## RESULTS
    print(f"#### FOLD {fold+1} OOF Accuracy = {np.max(history['val_lwlrap']
```

```
FOLD: 1
TRAIN: [0 2 3 4 5 6 7 9] VALID: [1 8]
Downloading data from https://github.com/Callidior/keras-applications/relea
ses/download/efficientnet/efficientnet-b2_weights_tf_dim_ordering_tf_kernel
s_autoaugment_notop.h5
31940608/31936256 [==============================] - 2s 0us/step
Model: "Efficientnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
```

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

```
gaussian_noise (GaussianNois (None, None, None, 3)     0
_________________________________________________________________
efficientnet-b2 (Model)      (None, None, None, 1408)  7768562
_________________________________________________________________
global_average_pooling2d (Gl (None, 1408)              0
_________________________________________________________________
dropout (Dropout)            (None, 1408)              0
_________________________________________________________________
dense (Dense)                (None, 24)                33816
=================================================================
Total params: 7,802,378
Trainable params: 7,734,810
Non-trainable params: 67,568
_________________________________________________________________
Epoch 1/25
64/64 - 76s - lwlrap: 0.2361 - loss: 0.3987 - val_lwlrap: 0.2777 - val_los
s: 0.3057
Epoch 2/25
64/64 - 41s - lwlrap: 0.4638 - loss: 0.2833 - val_lwlrap: 0.4265 - val_los
s: 0.3045
Epoch 3/25
64/64 - 43s - lwlrap: 0.6416 - loss: 0.2667 - val_lwlrap: 0.6994 - val_los
s: 0.2691
Epoch 4/25
64/64 - 39s - lwlrap: 0.7965 - loss: 0.2479 - val_lwlrap: 0.6349 - val_los
s: 0.3010
Epoch 5/25
64/64 - 42s - lwlrap: 0.8901 - loss: 0.2328 - val_lwlrap: 0.7569 - val_los
s: 0.2727
Epoch 6/25
64/64 - 41s - lwlrap: 0.9329 - loss: 0.2236 - val_lwlrap: 0.7595 - val_los
s: 0.2712
Epoch 7/25
64/64 - 41s - lwlrap: 0.9589 - loss: 0.2173 - val_lwlrap: 0.7768 - val_los
s: 0.2790
Epoch 8/25
64/64 - 41s - lwlrap: 0.9778 - loss: 0.2107 - val_lwlrap: 0.8488 - val_los
s: 0.2540
Epoch 9/25
64/64 - 40s - lwlrap: 0.9881 - loss: 0.2071 - val_lwlrap: 0.8287 - val_los
s: 0.2568
Epoch 10/25
64/64 - 41s - lwlrap: 0.9953 - loss: 0.2044 - val_lwlrap: 0.8692 - val_los
s: 0.2403
Epoch 11/25
64/64 - 40s - lwlrap: 0.9965 - loss: 0.2029 - val_lwlrap: 0.8593 - val_los
s: 0.2473
Epoch 12/25
64/64 - 41s - lwlrap: 0.9990 - loss: 0.2021 - val_lwlrap: 0.8607 - val_los
s: 0.2414
Epoch 13/25
64/64 - 41s - lwlrap: 0.9979 - loss: 0.2021 - val_lwlrap: 0.8755 - val_los
s: 0.2372
Epoch 14/25
64/64 - 40s - lwlrap: 1.0000 - loss: 0.2010 - val_lwlrap: 0.8689 - val_los
s: 0.2406
Epoch 15/25
64/64 - 40s - lwlrap: 0.9998 - loss: 0.2008 - val_lwlrap: 0.8616 - val_los
s: 0.2386
Epoch 16/25
64/64 - 40s - lwlrap: 1.0000 - loss: 0.2006 - val_lwlrap: 0.8720 - val_los
s: 0.2358
Epoch 17/25
oss: 0.2004 - val_lwlrap: 0.8716 - val_los
```

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

```
s: 0.2350
Epoch 18/25
64/64 - 43s - lwlrap: 1.0000 - loss: 0.2004 - val_lwlrap: 0.8766 - val_los
s: 0.2341
Epoch 19/25
64/64 - 41s - lwlrap: 0.9998 - loss: 0.2004 - val_lwlrap: 0.8737 - val_los
s: 0.2344
Epoch 20/25
64/64 - 40s - lwlrap: 1.0000 - loss: 0.2003 - val_lwlrap: 0.8755 - val_los
s: 0.2338
Epoch 21/25
64/64 - 40s - lwlrap: 1.0000 - loss: 0.2003 - val_lwlrap: 0.8757 - val_los
s: 0.2332
Epoch 22/25
64/64 - 40s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8721 - val_los
s: 0.2328
Epoch 23/25
64/64 - 40s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8722 - val_los
s: 0.2330
Epoch 24/25
64/64 - 41s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8716 - val_los
s: 0.2329
Epoch 25/25
64/64 - 40s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8711 - val_los
s: 0.2330
#### FOLD 1 OOF Accuracy = 0.877

FOLD: 2
TRAIN: [1 2 3 4 6 7 8 9] VALID: [0 5]
Model: "Efficientnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
gaussian_noise_1 (GaussianNo (None, None, None, 3)     0
_________________________________________________________________
efficientnet-b2 (Model)      (None, None, None, 1408)  7768562
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1408)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1408)              0
_________________________________________________________________
dense_1 (Dense)              (None, 24)                33816
=================================================================
Total params: 7,802,378
Trainable params: 7,734,810
Non-trainable params: 67,568
_________________________________________________________________
Epoch 1/25
64/64 - 63s - lwlrap: 0.2316 - loss: 0.3918 - val_lwlrap: 0.2719 - val_los
s: 0.3307
Epoch 2/25
64/64 - 37s - lwlrap: 0.4610 - loss: 0.2836 - val_lwlrap: 0.4360 - val_los
s: 0.3202
Epoch 3/25
64/64 - 36s - lwlrap: 0.6336 - loss: 0.2678 - val_lwlrap: 0.5709 - val_los
s: 0.2902
Epoch 4/25
64/64 - 37s - lwlrap: 0.7800 - loss: 0.2500 - val_lwlrap: 0.5720 - val_los
s: 0.3195
Epoch 5/25
64/64 - 40s - lwlrap: 0.8854 - loss: 0.2344 - val_lwlrap: 0.7012 - val_los
```

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

```
Epoch 6/25
64/64 - 36s - lwlrap: 0.9244 - loss: 0.2247 - val_lwlrap: 0.7973 - val_los
s: 0.2635
Epoch 7/25
64/64 - 38s - lwlrap: 0.9662 - loss: 0.2154 - val_lwlrap: 0.8217 - val_los
s: 0.2545
Epoch 8/25
64/64 - 35s - lwlrap: 0.9786 - loss: 0.2110 - val_lwlrap: 0.7925 - val_los
s: 0.2553
Epoch 9/25
64/64 - 35s - lwlrap: 0.9872 - loss: 0.2081 - val_lwlrap: 0.8124 - val_los
s: 0.2637
Epoch 10/25
64/64 - 37s - lwlrap: 0.9974 - loss: 0.2041 - val_lwlrap: 0.8502 - val_los
s: 0.2467
Epoch 11/25
64/64 - 37s - lwlrap: 0.9988 - loss: 0.2027 - val_lwlrap: 0.8672 - val_los
s: 0.2412
Epoch 12/25
64/64 - 36s - lwlrap: 0.9989 - loss: 0.2023 - val_lwlrap: 0.8668 - val_los
s: 0.2385
Epoch 13/25
64/64 - 38s - lwlrap: 0.9990 - loss: 0.2015 - val_lwlrap: 0.8880 - val_los
s: 0.2345
Epoch 14/25
64/64 - 35s - lwlrap: 0.9990 - loss: 0.2014 - val_lwlrap: 0.8737 - val_los
s: 0.2367
Epoch 15/25
64/64 - 37s - lwlrap: 1.0000 - loss: 0.2009 - val_lwlrap: 0.8842 - val_los
s: 0.2373
Epoch 16/25
64/64 - 37s - lwlrap: 0.9995 - loss: 0.2007 - val_lwlrap: 0.8892 - val_los
s: 0.2333
Epoch 17/25
64/64 - 37s - lwlrap: 0.9995 - loss: 0.2006 - val_lwlrap: 0.8834 - val_los
s: 0.2340
Epoch 18/25
64/64 - 37s - lwlrap: 1.0000 - loss: 0.2004 - val_lwlrap: 0.8947 - val_los
s: 0.2322
Epoch 19/25
64/64 - 35s - lwlrap: 1.0000 - loss: 0.2003 - val_lwlrap: 0.8934 - val_los
s: 0.2323
Epoch 20/25
64/64 - 37s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8955 - val_los
s: 0.2314
Epoch 21/25
64/64 - 35s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8911 - val_los
s: 0.2321
Epoch 22/25
64/64 - 36s - lwlrap: 0.9998 - loss: 0.2003 - val_lwlrap: 0.8903 - val_los
s: 0.2318
Epoch 23/25
64/64 - 35s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8884 - val_los
s: 0.2315
Epoch 24/25
64/64 - 36s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8878 - val_los
s: 0.2317
Epoch 25/25
64/64 - 36s - lwlrap: 1.0000 - loss: 0.2001 - val_lwlrap: 0.8891 - val_los
s: 0.2318
#### FOLD 2 OOF Accuracy = 0.896

FOLD: 3
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js [2 7]
```

```
Model: "Efficientnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
gaussian_noise_2 (GaussianNo (None, None, None, 3)     0
_________________________________________________________________
efficientnet-b2 (Model)      (None, None, None, 1408)  7768562
_________________________________________________________________
global_average_pooling2d_2 ( (None, 1408)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1408)              0
_________________________________________________________________
dense_2 (Dense)              (None, 24)                33816
=================================================================
Total params: 7,802,378
Trainable params: 7,734,810
Non-trainable params: 67,568
_________________________________________________________________
Epoch 1/25
64/64 - 66s - lwlrap: 0.2378 - loss: 0.3912 - val_lwlrap: 0.2526 - val_los
s: 0.3044
Epoch 2/25
64/64 - 39s - lwlrap: 0.4660 - loss: 0.2837 - val_lwlrap: 0.4712 - val_los
s: 0.3065
Epoch 3/25
64/64 - 39s - lwlrap: 0.6455 - loss: 0.2667 - val_lwlrap: 0.5528 - val_los
s: 0.3369
Epoch 4/25
64/64 - 38s - lwlrap: 0.7854 - loss: 0.2499 - val_lwlrap: 0.7572 - val_los
s: 0.2603
Epoch 5/25
64/64 - 40s - lwlrap: 0.8770 - loss: 0.2358 - val_lwlrap: 0.7756 - val_los
s: 0.2553
Epoch 6/25
64/64 - 37s - lwlrap: 0.9219 - loss: 0.2262 - val_lwlrap: 0.7663 - val_los
s: 0.2654
Epoch 7/25
64/64 - 39s - lwlrap: 0.9506 - loss: 0.2186 - val_lwlrap: 0.8637 - val_los
s: 0.2399
Epoch 8/25
64/64 - 37s - lwlrap: 0.9589 - loss: 0.2156 - val_lwlrap: 0.8482 - val_los
s: 0.2464
Epoch 9/25
64/64 - 38s - lwlrap: 0.9777 - loss: 0.2108 - val_lwlrap: 0.8646 - val_los
s: 0.2472
Epoch 10/25
64/64 - 39s - lwlrap: 0.9868 - loss: 0.2076 - val_lwlrap: 0.8786 - val_los
s: 0.2426
Epoch 11/25
64/64 - 37s - lwlrap: 0.9912 - loss: 0.2059 - val_lwlrap: 0.8779 - val_los
s: 0.2389
Epoch 12/25
64/64 - 37s - lwlrap: 0.9944 - loss: 0.2044 - val_lwlrap: 0.8730 - val_los
s: 0.2386
Epoch 13/25
64/64 - 39s - lwlrap: 0.9949 - loss: 0.2038 - val_lwlrap: 0.8894 - val_los
s: 0.2340
Epoch 14/25
64/64 - 37s - lwlrap: 0.9980 - loss: 0.2022 - val_lwlrap: 0.8874 - val_los
s: 0.2370
Epoch 15/25
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js oss: 0.2024 - val_lwlrap: 0.8846 - val_los
```

```
s: 0.2395
Epoch 16/25
64/64 - 37s - lwlrap: 0.9981 - loss: 0.2016 - val_lwlrap: 0.8813 - val_los
s: 0.2335
Epoch 17/25
64/64 - 38s - lwlrap: 0.9984 - loss: 0.2012 - val_lwlrap: 0.8939 - val_los
s: 0.2330
Epoch 18/25
64/64 - 38s - lwlrap: 0.9989 - loss: 0.2009 - val_lwlrap: 0.8862 - val_los
s: 0.2355
Epoch 19/25
64/64 - 37s - lwlrap: 0.9986 - loss: 0.2009 - val_lwlrap: 0.8829 - val_los
s: 0.2331
Epoch 20/25
64/64 - 37s - lwlrap: 0.9979 - loss: 0.2009 - val_lwlrap: 0.8897 - val_los
s: 0.2311
Epoch 21/25
64/64 - 36s - lwlrap: 0.9996 - loss: 0.2005 - val_lwlrap: 0.8894 - val_los
s: 0.2311
Epoch 22/25
64/64 - 37s - lwlrap: 0.9981 - loss: 0.2008 - val_lwlrap: 0.8933 - val_los
s: 0.2303
Epoch 23/25
64/64 - 36s - lwlrap: 0.9998 - loss: 0.2006 - val_lwlrap: 0.8884 - val_los
s: 0.2309
Epoch 24/25
64/64 - 38s - lwlrap: 0.9991 - loss: 0.2006 - val_lwlrap: 0.8882 - val_los
s: 0.2311
Epoch 25/25
Restoring model weights from the end of the best epoch.
64/64 - 38s - lwlrap: 0.9994 - loss: 0.2005 - val_lwlrap: 0.8865 - val_los
s: 0.2311
Epoch 00025: early stopping
#### FOLD 3 OOF Accuracy = 0.894

FOLD: 4
TRAIN: [0 1 2 3 5 6 7 8] VALID: [4 9]
Model: "Efficientnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
gaussian_noise_3 (GaussianNo (None, None, None, 3)     0
_________________________________________________________________
efficientnet-b2 (Model)      (None, None, None, 1408)  7768562
_________________________________________________________________
global_average_pooling2d_3 ( (None, 1408)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1408)              0
_________________________________________________________________
dense_3 (Dense)              (None, 24)                33816
=================================================================
Total params: 7,802,378
Trainable params: 7,734,810
Non-trainable params: 67,568
_________________________________________________________________
Epoch 1/25
64/64 - 68s - lwlrap: 0.2038 - loss: 0.3983 - val_lwlrap: 0.2198 - val_los
s: 0.3227
Epoch 2/25
64/64 - 40s - lwlrap: 0.3959 - loss: 0.2875 - val_lwlrap: 0.3728 - val_los
s: 0.2976
```

```
64/64 - 40s - lwlrap: 0.6115 - loss: 0.2697 - val_lwlrap: 0.5673 - val_los
s: 0.2972
Epoch 4/25
64/64 - 39s - lwlrap: 0.7623 - loss: 0.2524 - val_lwlrap: 0.6554 - val_los
s: 0.2851
Epoch 5/25
64/64 - 40s - lwlrap: 0.8374 - loss: 0.2410 - val_lwlrap: 0.6087 - val_los
s: 0.3124
Epoch 6/25
64/64 - 38s - lwlrap: 0.9064 - loss: 0.2300 - val_lwlrap: 0.7712 - val_los
s: 0.2618
Epoch 7/25
64/64 - 37s - lwlrap: 0.9428 - loss: 0.2221 - val_lwlrap: 0.7471 - val_los
s: 0.2857
Epoch 8/25
64/64 - 39s - lwlrap: 0.9633 - loss: 0.2161 - val_lwlrap: 0.8023 - val_los
s: 0.2766
Epoch 9/25
64/64 - 38s - lwlrap: 0.9729 - loss: 0.2117 - val_lwlrap: 0.8011 - val_los
s: 0.2607
Epoch 10/25
64/64 - 40s - lwlrap: 0.9834 - loss: 0.2088 - val_lwlrap: 0.8354 - val_los
s: 0.2636
Epoch 11/25
64/64 - 38s - lwlrap: 0.9886 - loss: 0.2063 - val_lwlrap: 0.8046 - val_los
s: 0.2613
Epoch 12/25
64/64 - 37s - lwlrap: 0.9926 - loss: 0.2049 - val_lwlrap: 0.8329 - val_los
s: 0.2613
Epoch 13/25
64/64 - 38s - lwlrap: 0.9946 - loss: 0.2042 - val_lwlrap: 0.8329 - val_los
s: 0.2555
Epoch 14/25
64/64 - 37s - lwlrap: 0.9945 - loss: 0.2032 - val_lwlrap: 0.8264 - val_los
s: 0.2535
Epoch 15/25
64/64 - 39s - lwlrap: 0.9968 - loss: 0.2026 - val_lwlrap: 0.8335 - val_los
s: 0.2607
Epoch 16/25
64/64 - 38s - lwlrap: 0.9968 - loss: 0.2018 - val_lwlrap: 0.8225 - val_los
s: 0.2572
Epoch 17/25
64/64 - 40s - lwlrap: 0.9954 - loss: 0.2020 - val_lwlrap: 0.8490 - val_los
s: 0.2500
Epoch 18/25
64/64 - 37s - lwlrap: 0.9987 - loss: 0.2013 - val_lwlrap: 0.8471 - val_los
s: 0.2499
Epoch 19/25
64/64 - 37s - lwlrap: 0.9969 - loss: 0.2015 - val_lwlrap: 0.8390 - val_los
s: 0.2489
Epoch 20/25
64/64 - 37s - lwlrap: 0.9974 - loss: 0.2012 - val_lwlrap: 0.8374 - val_los
s: 0.2477
Epoch 21/25
64/64 - 37s - lwlrap: 0.9969 - loss: 0.2013 - val_lwlrap: 0.8451 - val_los
s: 0.2457
Epoch 22/25
64/64 - 38s - lwlrap: 0.9980 - loss: 0.2011 - val_lwlrap: 0.8363 - val_los
s: 0.2480
Epoch 23/25
64/64 - 38s - lwlrap: 0.9985 - loss: 0.2008 - val_lwlrap: 0.8419 - val_los
s: 0.2467
Epoch 24/25
oss: 0.2009 - val_lwlrap: 0.8427 - val_los
```

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

```
s: 0.2465
Epoch 25/25
Restoring model weights from the end of the best epoch.
64/64 - 39s - lwlrap: 0.9987 - loss: 0.2010 - val_lwlrap: 0.8455 - val_los
s: 0.2464
Epoch 00025: early stopping
#### FOLD 4 OOF Accuracy = 0.849

FOLD: 5
TRAIN: [0 1 2 4 5 7 8 9] VALID: [3 6]
Model: "Efficientnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
gaussian_noise_4 (GaussianNo (None, None, None, 3)     0
_________________________________________________________________
efficientnet-b2 (Model)      (None, None, None, 1408)  7768562
_________________________________________________________________
global_average_pooling2d_4 ( (None, 1408)              0
_________________________________________________________________
dropout_4 (Dropout)          (None, 1408)              0
_________________________________________________________________
dense_4 (Dense)              (None, 24)                33816
=================================================================
Total params: 7,802,378
Trainable params: 7,734,810
Non-trainable params: 67,568
_________________________________________________________________
Epoch 1/25
64/64 - 63s - lwlrap: 0.2413 - loss: 0.3850 - val_lwlrap: 0.2220 - val_los
s: 0.3604
Epoch 2/25
64/64 - 37s - lwlrap: 0.4454 - loss: 0.2845 - val_lwlrap: 0.4904 - val_los
s: 0.2982
Epoch 3/25
64/64 - 37s - lwlrap: 0.6433 - loss: 0.2668 - val_lwlrap: 0.6935 - val_los
s: 0.2707
Epoch 4/25
64/64 - 37s - lwlrap: 0.8085 - loss: 0.2466 - val_lwlrap: 0.7082 - val_los
s: 0.2937
Epoch 5/25
64/64 - 39s - lwlrap: 0.8878 - loss: 0.2337 - val_lwlrap: 0.7374 - val_los
s: 0.2693
Epoch 6/25
64/64 - 37s - lwlrap: 0.9379 - loss: 0.2233 - val_lwlrap: 0.8211 - val_los
s: 0.2632
Epoch 7/25
64/64 - 38s - lwlrap: 0.9645 - loss: 0.2160 - val_lwlrap: 0.8354 - val_los
s: 0.2761
Epoch 8/25
64/64 - 35s - lwlrap: 0.9744 - loss: 0.2126 - val_lwlrap: 0.8288 - val_los
s: 0.2546
Epoch 9/25
64/64 - 37s - lwlrap: 0.9896 - loss: 0.2076 - val_lwlrap: 0.8404 - val_los
s: 0.2495
Epoch 10/25
64/64 - 38s - lwlrap: 0.9932 - loss: 0.2051 - val_lwlrap: 0.8616 - val_los
s: 0.2466
Epoch 11/25
64/64 - 37s - lwlrap: 0.9990 - loss: 0.2029 - val_lwlrap: 0.8724 - val_los
s: 0.2398
```

```
64/64 - 37s - lwlrap: 0.9989 - loss: 0.2021 - val_lwlrap: 0.8817 - val_los
s: 0.2445
Epoch 13/25
64/64 - 35s - lwlrap: 0.9998 - loss: 0.2012 - val_lwlrap: 0.8810 - val_los
s: 0.2352
Epoch 14/25
64/64 - 35s - lwlrap: 0.9998 - loss: 0.2010 - val_lwlrap: 0.8627 - val_los
s: 0.2407
Epoch 15/25
64/64 - 39s - lwlrap: 1.0000 - loss: 0.2007 - val_lwlrap: 0.8839 - val_los
s: 0.2344
Epoch 16/25
64/64 - 35s - lwlrap: 0.9998 - loss: 0.2008 - val_lwlrap: 0.8688 - val_los
s: 0.2369
Epoch 17/25
64/64 - 37s - lwlrap: 1.0000 - loss: 0.2005 - val_lwlrap: 0.8909 - val_los
s: 0.2360
Epoch 18/25
64/64 - 38s - lwlrap: 1.0000 - loss: 0.2003 - val_lwlrap: 0.8932 - val_los
s: 0.2334
Epoch 19/25
64/64 - 35s - lwlrap: 1.0000 - loss: 0.2003 - val_lwlrap: 0.8905 - val_los
s: 0.2321
Epoch 20/25
64/64 - 36s - lwlrap: 1.0000 - loss: 0.2003 - val_lwlrap: 0.8877 - val_los
s: 0.2329
Epoch 21/25
64/64 - 36s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8916 - val_los
s: 0.2319
Epoch 22/25
64/64 - 36s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8918 - val_los
s: 0.2319
Epoch 23/25
64/64 - 37s - lwlrap: 1.0000 - loss: 0.2002 - val_lwlrap: 0.8920 - val_los
s: 0.2319
Epoch 24/25
64/64 - 36s - lwlrap: 1.0000 - loss: 0.2001 - val_lwlrap: 0.8920 - val_los
s: 0.2318
Epoch 25/25
64/64 - 37s - lwlrap: 1.0000 - loss: 0.2001 - val_lwlrap: 0.8950 - val_los
s: 0.2317
```

Plot curve

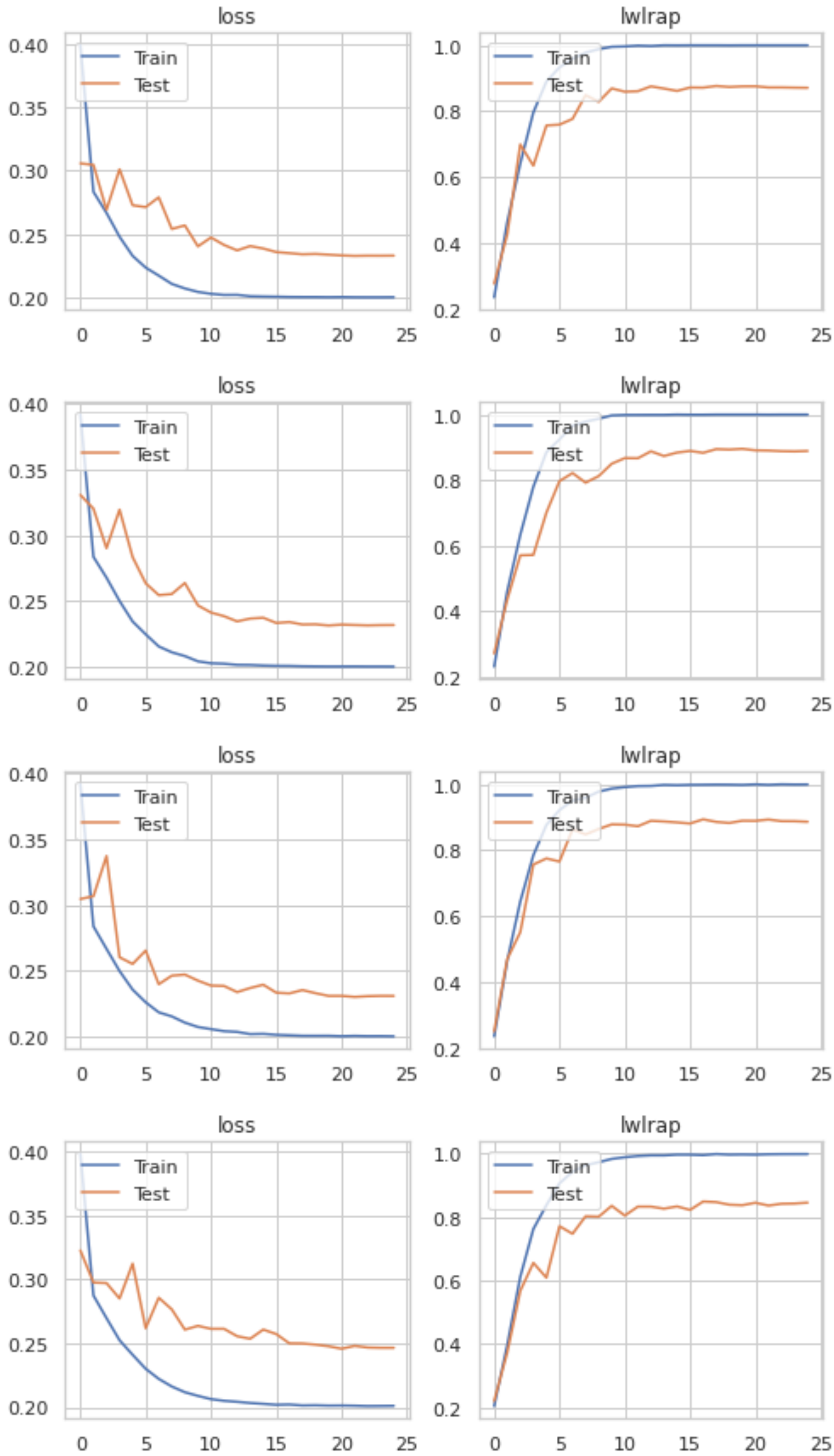
In [20]:

```
def plot_history(history):
    plt.figure(figsize=(8,3))
    plt.subplot(1,2,1)
    plt.plot(history["loss"])
    plt.plot(history["val_loss"])
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.title("loss")

    plt.subplot(1,2,2)
    plt.plot(history["lwlrap"])
    plt.plot(history["val_lwlrap"])
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.title("lwlrap")

for hist in history_list:
    plot_history(hist)
```

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


loss
lwlrap
Train
Test
0.40
0.35
0.30
0.25
0.20
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25

Inference

In [21]:
```
def get_test_dataset(filenames, training = False):

    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads = AUTO
    dataset = dataset.map(read_unlabeled_tfrecord , num_parallel_calls = AU
    dataset = dataset.map(lambda spec : waveform_to_log_mel_spectrogram(spe
    dataset = dataset.map(preprocess, num_parallel_calls = AUTO)
    return dataset.batch(GLOBAL_BATCH_SIZE*4).cache()
```

In [22]:
```
test_predict = []

test_data = get_test_dataset(TEST_FILES, training = False)
test_audio = test_data.map(lambda frames, recording_id: frames)

for fold in range(N_FOLDS):
    model.load_weights(f'./RFCX_model_fold {fold}.h5')
    test_predict.append(model.predict(test_audio, verbose = 1 ))
```

```
94/94 [==============================] - 108s 1s/step
94/94 [==============================] - 39s 415ms/step
94/94 [==============================] - 39s 416ms/step
94/94 [==============================] - 39s 416ms/step
94/94 [==============================] - 39s 416ms/step
```

Submission

In [23]:
```
np.array(test_predict).shape
```

Out[23]: (5, 11952, 24)

In [24]:
```
SUB = pd.read_csv('../input/rfcx-species-audio-detection/sample_submission

predict = np.array(test_predict).reshape(N_FOLDS, len(SUB), 60 // TIME, pa
predict = np.mean(np.max(predict ,axis = 2) , axis = 0)
# predict = np.mean(predict, axis =  0)

recording_id = test_data.map(lambda frames, recording_id: recording_id).unl
# # all in one batch
test_ids = next(iter(recording_id.batch(len(SUB) * 60 // TIME))).numpy().as

pred_df = pd.DataFrame({ 'recording_id' : test_ids[:, 0],
             **{f's{i}' : predict[:, i] for i in range(params.num_classes)}
```

In [25]:
```
pred_df.sort_values('recording_id', inplace = True)
pred_df.to_csv('submission.csv', index = False)
```

In [26]:
```
pred_df
```

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

Out[26]:

	recording_id	s0	s1	s2	s3	s4	s5	s6	
0	000316da7	0.064478	0.077867	0.061001	0.900531	0.092453	0.067719	0.083469	0.100
32	003bc2cb2	0.040720	0.046613	0.046107	0.115369	0.046783	0.048467	0.060647	0.052
64	0061c037e	0.152087	0.063591	0.077135	0.385517	0.079860	0.213214	0.065843	0.723
96	010eb14d3	0.962930	0.033194	0.044012	0.047191	0.042139	0.058357	0.043953	0.047
128	011318064	0.047873	0.048189	0.051123	0.547272	0.046555	0.054145	0.052897	0.054
...	...	...	...	...	...	...	...	...	
1119	ff68f3ac3	0.073262	0.050557	0.057655	0.239713	0.054878	0.881876	0.052771	0.280
1151	ff973e852	0.052483	0.052631	0.051936	0.051545	0.051859	0.102557	0.049150	0.949
1183	ffa5cf6d6	0.062393	0.222523	0.083916	0.581710	0.063623	0.235306	0.065999	0.561
1215	ffa88cbb8	0.067814	0.130323	0.062958	0.961310	0.052219	0.220686	0.054750	0.925
1247	ffda5d7b3	0.044873	0.048462	0.985962	0.051807	0.047871	0.049237	0.053757	0.059

1992 rows × 25 columns

In [ ]:

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