==> general.txt <==
https://www.kaggle.com/c/rfcx-species-audio-detection/discussion/220339

==> link.txt <==
https://www.kaggle.com/meaninglesslives/rfcx-minimal

==> githubs.txt <==
top5: https://github.com/kuto5046/kaggle-rainforest
top10: (https://github.com/zhanghang1989/ResNeSt)
top21: https://github.com/MPGek/mpgek-rfcx-species-audio-detection-public
top23: https://github.com/dathudeptrai/rfcx-kaggle
top27: https://github.com/mnpinto/dl_pipeline

==> ipynb.txt <==
top9: https://www.kaggle.com/cdeotte/rainforest-post-process-lb-0-970

https://www.kaggle.com/mehrankazeminia/lb-0-980-rainforest-comparative-method-part-a

top38:
    https://www.kaggle.com/ashusma/training-rfcx-tensorflow-tpu-effnet-b2 - that was a great starter and I was just doing edits of that kernel to move on
    https://www.kaggle.com/aikhmelnytskyy/resnet-tpu-on-colab-and-kaggle - that showed how you can train on colab as well

==> papers.txt <==
top9 https://www.sciencedirect.com/science/article/pii/S1574954120300637
top43: http://dcase.community/documents/challenge2020/technical_reports/DCASE2020_Chan_6.pdf

==> top1.txt <==
Thanks to Kaggle and hosts for this very interesting competition with a tricky setup. This has been as always a great collaborative effort and please also give your upvotes to @christofhenkel and @ilu000. In the following, we want to give a rough overview of our winning solution.
TLDR

Our solution is an ensemble of several CNNs, which take a mel spectrogram representation of the recording as input and predict on recording level using ¿weak labels¿ or on a more granular time level using ¿hard labels¿. Key in our modeling is masking as part of the loss function to only account for provided annotations. In order to account for the large amount of missing annotations and the inconsistent way how train and test data was labeled we apply a sophisticated scaling of model predictions.
Data setup & CV

As most participants know, the training data was substantially d

ifferently labeled compared to the test data and the training la bels were sparse. Hence, it was really tricky, nearly impossible to get a proper validation setup going. We tried quite a few th ings, such as treating all top 3 predicted labels as TPs when ca lculating the LWLRAP (because we know that on average a recordin g has 3 TPs), or calculating AUC only on segments where we know the labels (masked AUC), but in the end there was no good correl ation that we could find to the public LB. This meant that we ha d to fully rely on public LB as feedback for choosing our models and submissions. Thankfully, it was a random split from the ful l test population, but everything else would not have made much sense anyways most likely.

Models

Its worthy to note that for most models we performed also the me l spec transformation and augmentations like mixup or coarse dro pout on GPU using the implementation that can be found under tor chlibrosa (https://github.com/qiuqiangkong/torchlibrosa/blob/mas ter/torchlibrosa/stft.py).

Our final models incorporate both hard and weak label models as explained next.

Hard label models

We refer to hard labels as labels that have hard time boundaries inside the recordings. Our hard label models were trained on th e provided TPs (target = 1) and FPs (target = 0) labels with tim e aware loss evaluation. We used a log-spectrogram tensor of var iable time length as input to an EfficientNet backbone and restr icted the pooling layer to only mean pool over the frequency axi s. After pooling, the output has 24 channels for each species an d a time dimension.

We then map the time axis from the model to the time labels from the TPs and FPs and evaluate the BCE loss only for the parts wi th provided labels. For all other segments (which is actually th e majority) the loss is ignored, as we have no prior knowledge a bout the presence or absence of species there. In the figure bel ow we show how a masked label looks like: yellow means target=1, green is target=0 and purple is ignored.

For some models we added hand labeled parts of the train set but saw diminishing returns when labeling species that were missed by the TP/FP detector, which makes us wonder how the test labeli ng was done. Also, we wonder where the cut was made for backgrou nd songs (e.g. species 2 had some calls in the background of sev eral recordings, but the parts were labeled as FP). Most notably , adding TP labels for species 18 gave a substantial boost to LB score, and we believe that adding some hand labels to the mix o f models in the blend helped with diversity and generalization.

For some models, similar to other top performing teams, we train ed a second stage in which we replaced the masked part of the la bel with pseudo predictions of the first stage, but downweighted with factor 0.5. The main difference here to other teams is tha t we scaled the pseudo predictions in the same way we scale test predictions.

As augmentation we used mixup with lambda=3, SpecAugment and gaussian noise.

Weak label models

The models in this part of the blend are based on weak label models. The input is the log-spectrogram of the full 60 seconds of an audio recording including all the labels for that clip. So it directly fits on the format where the final predictions need to be made. Due to missing labels, just fitting on the known TPs does not work too well as we incorporate wrong labels by nature. Also we cannot use the FPs, because even though an FP might be present in one part of the recording, does not mean there might not be a TP at another position.

Hence, the models fit here include pseudo labels from our hard label models (see above) as well as some partial hand labels. For the pseudo labels, we take the raw output from the hard label models, but scale them to our expected true distribution (see post processing). For the hand labels, we only pick the TPs as well as FPs that span over a 60second period so that we are sure the species is not part of that recording. In loss, we weight the pseudos between 0.3-0.5 and the original labels and hand labels as 1.

If we would just fit on the raw pseudo outputs, we would not learn anything new, so we employ concepts from noisy-student models. That means we utilize not only simple augmentations and mixup, but also randomly sample pseudo labels for each recording each time we train on it based on a pool of stage 1 hard label models. So for example, you fit 10 hard label models, and then randomly sample one each time in the dataloader. This introduces randomness and further boosts on top of the stage 1 models.

Additionally, we fit several backbones (efnetb0, efnetb3, seresnext26, mobilenetv2_120d) where each is trained on the full data (no folds) with several seeds. In the end this part of the blend is a bag of around 120 models, where some also have additional TTA (horizontal flip).

How we are blending

We are blending different model types described above as depicted by the following graphic:

Post processing

We noticed that the test distribution of the target labels is substantially different to the provided train labels. Due to this fact, the models assume an unreasonable low or high probability when they are uncertain (Chris already has started a great thread about it here). To tackle this, we used several a priori information from the test distribution and scaled our predictions accordingly: by probing the public leaderboard we extracted a test label distribution which was aligning well with a previous research paper from the hosts. With additional prior knowledge about the average number of labels per row (3) -- also confirmed by LB probing, as well as the research paper -- we applied either a linear (species_probas *= factor) or a power scaling (species_probas **= factor) per species to our predicted probabilities to match the top3 predictions distribution (orange) with the previous

ly mentioned estimated test distribution (blue). But we didn¿t s
top there, as we know that the number of labels per row is not a
lways 3 but can be as low as 1 or as high as 8 (stated in the pa
per). Based on the sum of our probas in each row, we estimated t
he most likely topX (with a minimum count of 1) distribution (gr
een) of the test set, and optimized the scaling factors by minim
izing the total sum of the errors.
What did not work

I think in the end quite a few things we tried ended up in the b
lend fostering the diversity in it. But naturally, there are als
o many different things that did not work, after all we ran clos
e to 2,000 experiments throughout the course of this competition
. One noteworthy thing we tried was object detection based on th
e bounding boxes we had available in training. It worked reasona
bly well on simple CV setting reaching >0.7 LWLRAP on full 60 se
cond recordings, but we never continued to work on it on smaller
 crops or other settings.

We explored quite some architectures in the hope to improve our
ensemble. So we tried models that work on the raw wave like Res1
DNet or the just released wav2vec. But none did sufficiently wel
l.

Thanks for reading. Questions are very welcome.
Christof, Pascal & Philipp
========== a ====
without post processing, the score would be significantly lower.
 As stated here and also in other threads, there are a few ways
to tackle the class imbalance. One is post processing the distri
butions to match the expected test distibution, others include e
.g. adding labels for the underrepresented species as some other
 teams did. We believe, that all high scores include some sort o
f technique that draws the predictions closer to the expected te
st label distribution.
Also note, that the train label distribution, if it would includ
e all labels, should also be quite close to the test label distr
ibution.
=== q ==

Pseudo training and manaually debiasing the prediction seems to
be key part for this competition. Also your approach for the pro
blem with concept of hard/weak labels is great.
I see your team ensembled various models. If you have, can you s
hare the score of your best single model on lb?
=== q= ==
Giba ¿ (39th in this Competition) ¿ 2 days ago ¿ Options ¿ Repor
t ¿ Reply

4

Congrats @philippsinger @christofhenkel and @ilu000 for this gre
at insight and huge win!
Thanks for sharing the approach.
Did you guys tested LB score without testset distribution adjust
ment (PP) ?

Unfortunately one more competition with results heavily based in

LB probing and external data leakage.
=== a= ===
I agree, specifically that the paper exists is a bit weird, not the first time for research competitions that this happens.

Without the PP is not really possible for us as we already incorporate our pseudos this way and the final dist is already biased towards that. I think in the end the metric needs some form of scaling. For example, as the data contains 90% S3 labels, if you do not predict these high enough, then the metric is hurt a lot. But the scaling can be achieved via different things. For example if you hand-label all the data as some did then you automatically move towards the test distribution as the populations are roughly similar. I think ranking loss maybe has some potential, but we did not find time to explore it.

After all, I see the public dataset as a validation set here. And the validation set is a fair sample from the test set. So naturally you will try to fit the validation set better, which includes properly moving the TPs to the top.
=== a==
Yeah we also used heuristics to increase the more frequent classes. Pseudo labeling, mean max blending, handlabeling moved all to that direction.
We did not use LB probing this time because of lack of submissions and we were afraid of overfitting¿

It was surprising that even further scaling coulld boost our scores¿.

==> top2.txt <==

I trained simple classification models (24 binary classes) with logmel spectrograms :

- bootstrap stage: models are trained on TP/FP with masked BCE loss
- generate soft pseudo labels with 0.5 second sliding window
- train models with pseudo labels and also sample (with p=0.5) places with TP/FP - this partially solves confirmation bias problem.

Rounds of pseudo labeling and retraining (points 2,3) were repeated until the score on public LB didn't improve. Depending on the settings it took around 4-10 rounds to converge.

My initial models that gave 0.86 on TP/FP alone easily reached 0.96x with pseudo labeling . After this success I gave this challenge a 5 weeks break as I lost any motivation to improve my score :)
Later to my surprise it was extremely hard to beat 0.97 even with improved first stage models.
Melspectrogram parameters

- 256 mel bins
- 512 hop length

- original SR
- 4096 nfft

FreqConv (CoordConv for frequency)

After my first successful experiment with pseudo labeling that r eached 0.969 on public LB I tried to just swap encoders and blen d models but this did not bring any improvements.
So I visualised the data for different classes and understood th at when working with mel spectrograms for this task we don¿t nee d translation invariance and classes really depend on both frequ ency and patterns.
I added a channel to CNN input which contains the number of mel bin scaled to 0-1. This significantly improved validation metric s and after this change log loss on crops around TP after the fi rst round of training with pseudo labels was around 0.04 (same f or crops around FP). Though it only slightly improved results on the LB.
First stage

For the first stage I used all tp/fp information without any sam pling and made crops around the center of the signal.

Augmentations

- time warping
- random frequency masking below TP/FP signal
- random frequency masking above TP/FP signal
- gaussian noise
- volume gain
- mixup on spectrograms

For mixup on spectrograms - I used constant alpha (0.5) and hard labels with clipping (0,1). Masks were also added.
Pseudolabeling stages

Sampled TP/FP with p=0.5 otherwise made a random crop from the f ull spectrogram.

Without TP/FP sampling labels can become very soft and the score decreases after 2 or 3 rounds.
After training 4 folds of effnet/rexnet I generated OOF labels a nd ensembled their predictions. Then the training is repeated fr om scratch.

Augmentations

- gaussian noise
- volume gain
- mixup
- time warping
- spec augment
- mixup on spectrograms

Mixup

I used constant alpha (0.5) and added soft labels from two sampl es. This hurts logloss on FP a bit but at the same time signific

antly increases recall on TP.
Validation

Local validation did not have high correlation with the public l
eaderboard. Logloss on TP was somehow correlated but still it wa
s not robust.
So without proper validation I decided to not select the best ch
eckpoints and just trained 60 epochs (around 200 batches in each
 epoch) with CosineLR and AdamW optimizer.

My best models on validation - auc 0.999, log loss 0.03 did not
produce great results (0.95). After the competition though It tu
rned out that they can be easily improved with postprocessing to
 97x-98x range.
Final ensemble

I used 4 models with 4 folds from Effnet and Rexnet (https://arx
iv.org/abs/2007.00992 lightweight models with great performance)
 families:

    Rexnet-200 (4 sec training/inference), EffnetB3 (4 sec train
ing/inference)
    Rexnet-150 (8 sec training/inference), EffnetB1 (8 sec train
ing/inference)

Rexnet was much better than EfficientNet alone (less overfitting
), but in ensemble they worked great.

During inference I just used 0.5 second sliding window and took
max probabilities for the full clip and then averaged prediction
s from different models.
Lessons learned

I did not know about the paper and lacked this useful informatio
n about the dataset.

In my solutions I often rely on models alone but don¿t explore t
he data deeply.
In this case I understood that the relabeled train set has simil
ar class distribution to the test set and decided that models wo
uld easily learn that. I was wrong and simple post-processing co
uld significantly improve results (though this happened due to s
evere class imbalance).
==========
Did you try to add freqs to the loss function (it could force th
e network to use the FreqConv)?
I used CoorConv in some tasks and without adding the same coordi
nates to the loss function it didn't improve the network.
I have a simple check for it - pass 0 or noise to the channel wi
th coordinates, when I didn't add coordinates to the loss functi
on, it performs the same as with proper data in the CoordConv ch
annel. When I trained with coordinates in the loss, 0 or noise i
n the CoordConv - network inferences much worse.
==========a ==========
Good point! Just checked some checkpoint

With proper frequencies
neg_logloss: 0.1588028629548308

```
pos_logloss: 0.05171160377776966

With zeros
neg_logloss: 0.19649322897329777
pos_logloss: 0.3455986050609499

So my models really use it somehow
========== q ==========
Thanks for sharing your work. i am curious to learn new thing fr
om your work
i have a question on masked BCE loss. how is this implemented. w
hen you say mask, are you masking the loss for other classes tha
t are not in the label or masking the time frames(say 4sec input
 where the label is only for sec1-2) where there is no label.

second question . how are your using both TP and FP. how this lo
ss fn will look like.
===a ===
Masks for loss function have the same shape as labels, 1 for the
 classes we know (TP/FP) 0 for others

class BCEMasked(nn.Module):
   def forward(self, inputs, targets, mask=None):
       bce_loss = binary_cross_entropy_with_logits(inputs, targe
ts, reduction='none')
       if mask is not None:
           bce_loss = bce_loss[mask > 0]
       return bce_loss.mean()
========== q =========
           thanks for answering . about TP/FP , how are you usin
g FP for training the model. with sigmoid you cant have FP = -1.
========  a ==========
example - FP for s0
mask=[1, 0, 0, ¿,0], targets=[0, 0, 0, ¿, 0] - only the first el
ement from the targets and output will be considered by the loss
 function
======== q= ====
a question on this, do you train on the same random sampled audi
o crop and pseudo labels after or you re-sample tp/fp or made a
random crop again in stage 2?
====  a ====
My pseudolabels (for 4 second models) had 113 frames per audio c
lip (0.5 sec sliding window).
If TP/FP is sampled - I made a random crop around the center and
 then found nearest frame from pseudolabels. Pseudo labels were
fixed using tp/fp.
Otherwise I just took random frame from 113 and postprocessed la
bels if they overlap with tp/fp data.
=== q ==Rexnet-200 for 60 epochs.. wow)
=== a == https://arxiv.org/abs/2007.00992 Rexnet-200 (200 = 2.0
scale, 150 = 1.5) is a lightweight model. It is not related to r
esnet, it is a modification of MobileNet that performs like EffN
et.
So it was around 1 minute per epoch

==> top3.txt <==
3rd Place Solution
```

TLDR

Our solution is a mean blend of 8 models trained on True positive labels of all recording ids in train_tp.csv (given + hand-labeled labels) also from some recording ids in train_fp.csv (hand-labeled labels) with heavy augmentations. Additionally, some models are also trained on pseudo labels and a hand-labeled external dataset. We also post-processed the blended results by thresholding species 3.

From 308 submissions it is obvious that we have tested a lot of techniques and I will share more detailed information by category below.

Data Preparation

I couldn't get a proper validation framework setup after trying out many techniques and decided at one point to start digging into the data and figured out that there are many unlabeled samples both in and out of the range of t_min and t_max labels given in train_tp.csv. In https://www.kaggle.com/c/rfcx-species-audio-detection/discussion/197735 it was mentioned that using hand-labeled species was allowed, so I started labeling the data manually and after labeling 100 recording ids I could already get a > 0.9 public lb score and pretty consistent local CV scores that somewhat correlates with the public lb. Naturally, I continued to label the entire train_tp.csv seeing that it has only around 1.3k recording ids. Further labeling of train_fp.csv helped the score but only minimally so I stopped at one point. As I grew more familiar with the data I could label 300 recording ids in a day :), referring to pseudo labels helped a lot too. I also went through the train_tp.csv a few more rounds to make sure I have quality data. I used both spectrograms and listening strategy to analyze and label the data, some species are easy to spot with spectrograms and some are easier to spot by listening, and in some cases, both listening and visual inspection of the spectrograms can act as a multi verification technique to get more quality labels, especially when birds/frogs are very distant away from the recorder or there are strong noises like waterfall sounds. By labeling and analyzing the data I also figured out the kinds of sounds/noises that would appear and inspired me to try out a few augmentation methods which I will share below. Along with true positive labels, I also added noisy and non-noisy labels based on my confidence in the completeness of labels in a specific recording id. I am not a perfect labeler so I wanted to handle complete and non-complete labeled recording ids differently, which I will share below too.

I also removed some labels from train_tp.csv as I found some true positives suspicious, I didn't test not removing the labels before so not sure how much this helped.

Additionally, after finding out the paper from the organizers I searched for suitably licensed datasets with those species and found one dataset with species in this competition with a proper license. But there weren't any labels so I labeled it manually too with the same format as train_tp.csv. https://datadryad.org/stash/dataset/doi:10.5061/dryad.c0g2t . I reuploaded the dataset HERE with my manual labels.

I uploaded the extra labels as a dataset https://www.kaggle.com/dicksonchin93/extra-labels-for-rcfx-competition-data, feel free to use it and see if you can get a better single model score! mine was 0.970 on public lb

Modeling / Data Pre-processing

I used Mel Spectrograms with the following parameters: 32kHz sampling rate, a hop size of 716, a window size of 1366, and 224 or 128 Number of Mels. Tried a bunch of methods but plainly using 3 layers of standardized Mel Spectrograms works the best. The image dimensions were (num_mel_bins, 750).

Using train_tp.csv to create folds will potentially leak some training data into your validation data so I treated the problem as a multilabel target and used iterative-stratification to stratify the data into 5 partitions using unique recording ids and its multilabel targets. I had two different 5 fold partitions using different versions of the multi labels and used a mix of both in the final submission.

I used multiple different audio duration during the competition and at different stages of the competition, the best duration varied in my implementation but in the end, I used 5 seconds of audio for training and prediction as the LWLWRAP score was better on both public lb and local validation.

The 5-second audio was randomly sampled during training and in prediction time a 5-second sliding window was used with overlap and the max of predictions was used. How the 5-second audio is randomly sampled is considered to be an augmentation method in my opinion and so I will explain it in the heavy augmentations category below

Augmentations

Random 5-second audio samples:
a starting point was chosen randomly on values between reference t_mins and t_maxes obtained from

```
def get_ref_tmin_tmax_and_species_ids(
    self, all_tp_events, label_column_key="species_id"
):
        all_tp_events["t_min_ref"] = all_tp_events["t_min"].apply(
            lambda x: max(x - (self.period / 2.0), 0)
        )
        def get_tmax_ref(row, period=self.period):
            tmin_x = row["t_min"]
            tmax_x = row["t_max"]
            tmax_ref = tmax_x - (period / 4.0)
            if tmax_ref < tmin_x:
                tmax_ref = (tmax_x - tmin_x) / 2.0 + tmin_x
            return tmax_ref
        all_tp_events["t_max_ref"] = all_tp_events[
            ["t_max", "t_min"]
        ].apply(get_tmax_ref, axis=1)
        t_min_maxes = all_tp_events[
            ["t_min_ref", "t_max_ref"]
```

```
        ].values.tolist()
        species_ids = all_tp_events[label_column_key].values.tol
ist()
        return t_min_maxes, species_ids
```

Labels were also assigned based on the chosen starting time and ending time with t_min and t_max labels.

audio based pink noise
audio based white noise
reverberation
time stretch
use one of 16kHz or 48kHz sample rate data and resample it t o 32kHz sample rate using randomly chosen resampling methods ['k aiser_best', 'kaiser_fast', 'fft', 'polyphase']
use different window types to compute spectrograms at train time ['flattop', 'hamming', ('kaiser', 4.0), 'blackman', 'hann'] , hann window is used at test and validation time
masking out non labeled chunks of the audio with a 10% chanc e
one of spectrogram FMix and audio based mixup with the max o f labels instead of using the blend from the beta parameter
spec mix :
only one strip was used for each axis, for the horizontal ax is when the chosen frequency range to mask out completely covers a specific species minimum f_min and maximum f_max , that speci es label will be dropped. Specmix is also using the max of label s instead of using the blend from the beta parameter. The code b elow shows how I obtain the function that can output frequency a xis spectrogram positions from frequency

```
def get_mel_scaled_hz_to_y_axis_func(fmin=0, fmax=16000, n_mels=
128):
    hz_points = librosa.core.mel_frequencies(n_mels=n_mels, fmin
=fmin, fmax=fmax)
    hz_to_y_axis = interp1d(hz_points, np.arange(n_mels)[::-1])

    # reversed because first index is at the top left in an imag
e array
    return hz_to_y_axis
```

bandpass noise
Water from Freesound50k removing samples that have license t o prevent derivative work
Engine and Motor Sounds from Freesound50k removing samples t hat have license to prevent derivative work
Honk, Traffic and Horn sounds from Freesound50k removing sam ples that have license to prevent derivative work
Speech sounds from Freesound50k removing samples that have l icense to prevent derivative work
Bark sounds from Freesound50k removing samples that have lic ense to prevent derivative work

checkout recording_id b8d1e4865 to find dogs barking and some hu man speech :D
Architectures used

No SED just plain classifier models with GEM pooling for CNN bas

ed models

    Efficientnet-b7
    Efficientnet-b8
    HRNet w64
    deitbase224
    vit_large_patch16_224
    ecaresnet50
    2x resnest50 from https://www.kaggle.com/meaninglesslives, c
heckout his writeup in a minimal notebook HERE!

Loss

The main loss strategy used for the final submission was using d
ifferent loss function for samples which I am confident is compl
ete in labels and samples which I am not confident is complete i
n labels. BCE was used for non-noisy/confident samples and a mod
ified Lsoft loss was used on the noisy/non-confident. Lsoft loss
 was modified to be applied only to nonpositive samples, as I wa
s confident in my manual labels. It looks like this

```
def l_soft_on_negative_samples(y_pred, y_true, beta, eps = 1e-7)
:
    y_pred = torch.clamp(y_pred, eps, 1.0)

    # (1) dynamically update the targets based on the current st
ate of the model:
    # bootstrapped target tensor
    # use predicted class proba directly to generate regression
targets
    with torch.no_grad():
        negative_indexes = (y_true == 0).nonzero().squeeze(1)
        y_true_update = y_true
        y_true_update[negative_indexes[:, 0], negative_indexes[:
, 1]] = (
            y_true_update[negative_indexes[:, 0], negative_index
es[:, 1]] * beta +
            (1 - beta) * y_pred[negative_indexes[:, 0], negative
_indexes[:, 1]]
        )

    # (2) compute loss as always
    loss = F.binary_cross_entropy(y_pred, y_true_update)
    return loss
```

This was inspired by the first placed winner in the Freesound co
mpetition https://github.com/lRomul/argus-freesound but I notice
d that it doesn't make sense if it is used with mixup since audi
o will be mixed up anyways. So I also obtain the max of noisy bi
nary labels so that noisy labels mixed with clean labels are con
sidered to be noisy labels.
Pseudo Labels

I didn't get much boost from pseudo labels, maybe I did somethin
g wrong but nonetheless, it was used in some models. I used a 0.
8 threshold for labels generated with 5-second windows and utili
zed the same window positions during training. Using raw predict
ions didn't help the model at all on lb.

Post processing

We set the species 3 labels to be 1 with a 0.95 threshold and it boosted the score slightly

Other stuff

- Early stopping of 20 epochs with a minimum learning rate of 9e-6 to start counting these 20 epochs
- Reduce learning rate on Plateau with a factor of 0.6 and start with a few warmup epochs, when LR is reduced the best model weights was loaded back again

Things that failed

- using models without pre-trained weights
- timeshift
- using species from the Cornell Competition that are confused with species in this competition as a distractor noise, for example, moudov is similar to species 15, reevir is similar to species 11, rebwoo is similar to species 6, bkbwar is similar to species 7, cacwre is similar to species 19, every is similar to species 17 and nrwswa is similar to species 20
- using plane sounds from Freesound50k data
- using PCEN, deltas or CQT
- Random Power
- TTA with different window types
- Manifold mixup with resnest50
- using trainable Switchnorm as an initial layer replacing normal standardization
- using trainable Examplar norm as an initial layer replacing normal standardization
- Context Gating
- split audio intro three equal-length chunks and concat as 3 layer image
- lsep and Assymetric loss
- using rain sounds from Freesound50k data
- using a fixed validation mask similar to how I used random training mask
- use SIREN layer
- Tried to separate some confusing patterns as separate manual labels but didn't get the chance to test them

Hopefully, I didn't miss anything. Oh, we were holding off submitting a good model until @cpmpml came along :)

==> top4.txt <==

First of all, I'd like to thank my teammates, Rainforest Connection and Kaggle for this interesting and tricky challenge!

The major issue in this competition was obviously the labelling quality. True- and False-Positives audios contain lots of unlabelled regions that adds too much noise for the models. As a consequence, till the very end of the competition we haven't managed to establish a reliable local validation strategy and were mostly relying on the Public LB scores.

Moreover, labeled regions in the TP audios were balanced, i.e. e

ach class had an equal number of labels. However, we've noticed that test predictions contain mostly the 3rd class as a top-1 probability. And with the higher percentage of the 3rd class, the LB score tends to be better. The similar situation was for some other classes (e.g. top-2 was mostly the 18th class). It gave us an idea that probably test files have completely different class distributions compared to the TP data. That's why we've applied additional multipliers for the 3rd and 18th class to artificially increase probabilities for them (naming it class balancing).

Our final solution consists of 3 stages.
1st Stage

- Data: only TP labels on 26 classes (for each song_type).
- Models: SED-classifiers (EfficientNet-B1 and EfficientNet-B3)
- Cropping strategy: Random crops around TP regions
- Loss: BCE
- Augmentations: spectrogram augmentations (SpecAugment, Noise) and CutMix: cutting the TP regions and pasting them into the random time regions in the other TP and FP audios.
- Public LB score: 0.909 -> 940 (after balancing)
- Private LB score: 0.915 -> 0.938

2nd Stage

Taking the models from the 1st Stage we've made a set of pseudolabels for TP (OOF), FP and test data. The pseudolabels have been generated using the SED framewise output. At this point, audio files have much more labeled regions compared to the initial TP data. And on this stage models are being trained on the pseudolabels only. We've applied two approaches:
SED-classification

- Data: TP pseudolabels + random 2000 samples from FP pseudolabels for each fold. Use soft labels (0.9) for the pseudolabels
- Models: SED-classifiers (EfficientNet-B0, EfficientNet-B1, MobileNetV2, DenseNet121)
- Cropping strategy: Random 5 seconds crops around pseudolabeled regions
- Loss: modified LSEP loss
- Augmentations: raw audio augmentations, such as: GaussianNoiseSNR, PinkNoiseSNR, PitchShift, TimeShift, VolumeControl
- TTA: 6 different crop sizes are used during the inference: 2, 5, 10, 20, 30 and 60 seconds
- Best single model (5 fold) public LB score: 0.957 (after balancing)
- Private LB score: 0.963

Usual classification

- Data: TP + FP pseudolabels. Pre-train models on the test pseudolabels
- Models: Usual classifiers (EfficientNet-B1, ResNet34, SE-ResNeXt50)
- Cropping strategy: Random crops around pseudolabeled regions
- Loss: BCE
- Augmentations: spectrogram augmentations (SpecAugment, Noise

) and CutMix
    Best single model (5 fold) public LB score: 0.952 (after balancing)
    Private LB score: 0.959

3rd Stage

Taking the overall ensemble from the 2nd Stage allows to get the Public LB score of 0.965 (Private LB: 0.969). To achieve our best 0.969 Public LB (Private LB: 0.971) we're applying single class semantic segmentation models for 3rd, 11th and 18th classes (other classes didn't give any score improvements on the Public LB).

The segmentation polish is done in the following manner:
class_score = class_score * (1 + 0.1 * num_instances) if num_instances > 0 else class_score * 0.9, where num_instances is the number of instances predicted by the semantic segmentation model for each recording.
What didn`t work

    PANN pretrained weights (or other audio pretrained models) - imagenet performs best
    Using "fat" encoders
    Focal loss with soft penalty (But as we see It works for the other participants)
    Multiclass segmentation
    Raw audio classification with 1d convolutions

==> top5.txt <==
5th place solution (Training Strategy)

Congratulations to all the participants, and thanks a lot to the organizers for this competition! This has been a very difficult but fun competition:)

In this thread, I introduce our approach about training strategy.
About ensemble part will be written by my team member.

Our team ensemble each best model.

My model is Resnet18 which has a SED header. This model's LWLRAP is Public LB=0.949 /Private LB=0.951, and I trained by google colab using Theo Viel's npz dataset(32 kHz, 128 mels). Thank you, Theo Viel!!

Our approach has 3 stage,
Other team members are different in some things likes the base model and hyperparameter, but these default strategies are about the same.
1st stage: pre-train

¿I think this part is not important. Team member Ahmet skips this part.

This stage transfers learning from Imagenet to spectograms.

Theo Viel's npz dataset can be regarded as 128x3751 size image.
I cut to 512 by this image in sound point from t_min and t_max.
I train this image by tp_train and 30 sampled fp_train.

Parameters:

    Adam
    learning_rate=1e-3
    CosineAnnealingLR(max_T=10)
    epoch=50

Continue 2nd and 3rd stage use this trained weight.
2nd stage: pseudo label re-labeling

The purpose of stage2 is to improve the model and make pseudo la
bels by this model.

Use 1st stage trained weight.

The key point I think is to calculate gradient loss only labeled
 frame. The positive labels were sampled from tp_train.csv only
and the negative labels were sampled from fp_train.csv only.
I put 1 to positive label and -1 to negative label.

```
tp_dict = {}
for recording_id, df in train_tp.groupby("recording_id"):
    tp_dict[recording_id+"_posi"] = df.values[:, [1,3,4,5,6]]

fp_dict = {}
for recording_id, df in train_fp.groupby("recording_id"):
    fp_dict[recording_id+"_nega"] = df.values[:, [1,3,4,5,6]]

def extract_seq_label(label, value):
    seq_label = np.zeros((24, 3751))  # label, sequence
    middle = np.ones(24) * -1
    for species_id, t_min, f_min, t_max, f_max in label:
        h, t = int(3751*(t_min/60)), int(3751*(t_max/60))
        m = (t + h)//2
        middle[species_id] = m
        seq_label[species_id, h:t] = value
    return seq_label, middle.astype(int)

# extract positive label and middle point
fname = "00204008d" + "_posi"
posi_label, posi_middle = extract_seq_label(tp_dict[fname], 1)

# extract negative label and middle point
fname = "00204008d" + "_nega"
nega_label, nega_middle = extract_seq_label(fp_dict[fname], -1)
```

loss function is that:

```
def rfcx_2nd_criterion(outputs, targets):
    clipwise_preds_att_ti = outputs["clipwise_preds_att_ti"]
    posi_label = ((targets == 1).sum(2) > 0).float().to(device)
    nega_label = ((targets == -1).sum(2) > 0).float().to(device)
```

```
    posi_y = torch.ones(clipwise_preds_att_ti.shape).to(device)
    nega_y = torch.zeros(clipwise_preds_att_ti.shape).to(device)
    posi_loss = nn.BCEWithLogitsLoss(reduction="none")(clipwise_
preds_att_ti, posi_y)
    nega_loss = nn.BCEWithLogitsLoss(reduction="none")(clipwise_
preds_att_ti, nega_y)
    posi_loss = (posi_loss * posi_label).sum()
    nega_loss = (nega_loss * nega_label).sum()
    loss = posi_loss + nega_loss
    return loss
```

And image are cut and stack by sliding window.
I set the window size to 512 and cut out the entire range of 60' s audio data by covering it little by little. Cover 49 pixels ea ch, considering that important sounds may be located at the boun daries of the division.

```
N_SPLIT_IMG = 8
WINDOW = 512
COVER = 49

slide_img_pos = [[0, WINDOW]]
for idx in range(1, N_SPLIT_IMG):
    h, t = slide_img_pos[idx-1][0], slide_img_pos[idx-1][1]
    h = t - COVER
    t = h + WINDOW
    slide_img_pos.append([h, t])

print(slide_img_pos)
# [[0, 512], [463, 975], [926, 1438], [1389, 1901], [1852, 2364]
, [2315, 2827], [2778, 3290], [3241, 3753]]
```

I predict each sliding window and put the pseudo label, so I got 8 windows in one 60 sec recording.

patch idx	pixcel	time(s)
0	0¿512	0¿8
1	463¿975	7¿15
2	926¿1438	14¿23
3	1389¿1901	22¿30
4	1852¿2364	29¿37
5	2315¿2827	37¿45
6	2778¿3290	44¿52
7	3241¿3753	51¿60

Parameters:

```
    Adam
    learning_rate=3e-4
    CosineAnnealingLR(max_T=5)
    epoch=5
```

3rd stage: train by label re-labeled

This stage trains on the new labels re-labeled by 2nd stage mode l.

Use 1st stage trained weight.

The new label is ensemble by our team output like my 2nd stage.

    our prediction average value is
        >0.5: soft positive = 2
        <0.01: soft negative = -2

In this stage, I calculate gradient loss only labeled frame as w
ith 2nd stage.

Parameters:

    Adam
    learning_rate=3e-4
    CosineAnnealingLR(max_T=5)
    epoch=5

Some My Tips:

    Don't use soft negative.
    The re-label's loss(soft positive) is weighted 0.5.
    last layer mixup(from this blog)

CV

I use iterative-stratification's MultilabelStratifiedKFold. Vali
dation data is made from tp_train only and fp_train data is used
 training in all fold.

Each stage LWLRAP is that:

stage	CV	Public	Private
1st	0.7889	0.842	0.865
2nd	0.7766	0.874	0.878
3rd	0.7887	0.949	0.951

3rd stage's re-labeled LWRAP is 0.9621.
predict

In test time, I increase COVER to 256, so I got 14 windows in on
e 60 sec recording.
The prediction is max pooling in each patch.

I use clipwise_output in training, and I use framewise_output in
 prediction. This approach came from shinmura0's discussion thre
ad. Thank you shinmura0:)
did not work for my model

    TTA
    26 classes (divide song_type)
    label wright loss
    label smoothing(but team member's kuto improved)

Finally, I would like to thank the team members.
If I was alone, I couldn't get these result.
kuto, Ahmet, thank you very much.

My code:
https://github.com/trtd56/RFCX

It was important to have robust pseudolabels. Therefore, I have trained a Vision Transformer model and a Wavenet over resnet features for each class independently. Spectograms were generated with respect to min-max frequencies for each class. I have used the same architectures for training the last stage models on pseudolabels as well. Then I have aligned my logits to have the same mean and std as Toda's and got the optimal ensembling weights based on individual class AUC (tp vs not tp). This ensemble later is blended with Kuto's submission. Each model has different bias and training scheme. This way we achieved robustness.

=== q= ===
Congratulations for your great result!
I see you've got impressive boost from pseudo training (0.878->0.951). I have some questions. you said,

>0.5: soft positive = 2
<0.01: soft negative = -2

and

Don't use soft negative.

Do you mean when pseudo labeling, you converted >0.5 predictions to 1 and others to unknown(to mask out during loss computation)?
=== a ===
Thank you comment and sorry for late.

I treat separately original labels and pseudo labels.

My first 3rd stage loss function is that:

```
def rfcx_3rd_criterion(outputs, targets):
    clipwise_preds_att_ti = outputs["clipwise_preds_att_ti"]

    posi_label = ((targets == 1).sum(2) > 0).float().to(device)
    soft_posi_label = ((targets == 2).sum(2) > 0).float().to(device)
    nega_label = ((targets == -1).sum(2) > 0).float().to(device)
    soft_nega_label = ((targets == -2).sum(2) > 0).float().to(device)

    posi_y = torch.ones(clipwise_preds_att_ti.shape).to(device)
    nega_y = torch.zeros(clipwise_preds_att_ti.shape).to(device)

    posi_loss = nn.BCEWithLogitsLoss(reduction="none")(clipwise_preds_att_ti, posi_y)
    nega_loss = nn.BCEWithLogitsLoss(reduction="none")(clipwise_preds_att_ti, nega_y)
    soft_posi_loss = nn.BCEWithLogitsLoss(reduction="none")(clipwise_preds_att_ti, posi_y)
    soft_nega_loss = nn.BCEWithLogitsLoss(reduction="none")(clipwise_preds_att_ti, nega_y)
```

```
    posi_loss = (posi_loss * posi_label).sum()
    nega_loss = (nega_loss * nega_label).sum()
    soft_posi_loss = (soft_posi_loss * soft_posi_label).sum()
    soft_nega_loss = (soft_nega_loss * soft_nega_label).sum()

    loss = posi_loss + nega_loss + soft_posi_loss*0.5 + soft_neg
a_loss*0.5
    return loss
```

But soft_nega is not good work, so I have removed it.

==> top6.txt <==
42

It was a great competition. I'll document our journey primarily from my perspective, my teammates @jpison @amezet @pavelgonchar may chime in to add extra color¿

Our solution is rank ensemble of two types of models, primarily 97% using the architecture described below, and 3% an ensemble of SED models. The architecture below was made in the last 7 days ¿ I made my first (pretty bad) sub 8 days ago, and I joined team just before team merge deadline and all the ideas were done/implemented in ~7 days¿

A single 5-fold model achieves 0.961 private, 0.956 public; details:

Input representation. This is probably the key, we take each TP or FP and after spectrogram (not MEL, b/c MEL was intended to model human audition and the rainforrest species have not evolved like our human audition) crop it time and frequency wise with fixed size on a per-class basis. E.g. for class 0: minimum freq is 5906.25 Hz and max is 8250 Hz, and time-wise we take the longest of time length of TPs, i.e. for class 0: 1.29s.

With the above sample the spectrogram to yield an image, e.g.:

The GT for that TP would be:

```
tensor([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, n
an, nan, nan, 1., nan, nan, nan,nan, nan, nan, nan, nan, nan, na
n, nan]))
```

As other competitors, we expand the classes from 24 to 26 b/c to split species that have two songs.

Other sample:

```
tensor([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, n
an, 1., nan, nan, nan, nan, nan,nan, nan, nan, nan, nan, nan, na
n, nan]))
```

Note the image size is the same, so time and frequency are effectively strectched wrt to what the net will see, but I believe this is fine as long as the net has enough receptive field (which they have).

The reason of doing it this way is that we need to inject the time and frequency restrictions as inductive bias somehow, and this looks like an nice way.

Model architecture. This is just an image classifier outputing 26 logits, that's it. The only whistle is to add relative positional information in the freq axis (à la coordconv), so model is embarrasingly simple:

```
class TropicModel(Module):
    def __init__(self):
        self.trunk = timm.create_model(a.arch,pretrained=True,num_classes=n_species,in_chans=1+a.coord)
        self.do = nn.Dropout2d(a.do)
    def forward(self,x):
        bs,_,freq_bins,time_bins = x.size()
        coord = torch.linspace(-1,1,freq_bins,dtype=x.dtype,device=x.device).view(1,1,-1,1).expand(bs,1,-1,time_bins)
        if a.coord: x = torch.cat((x,coord),dim=1)
        x = self.do(x)
        return self.trunk(x)
```

Loss function. Just masked Focal loss. Actually this was a mistake b/c Focal loss was a renmant of a dead test and I (accidentally) left it there, where I thought (until I checked code now to write writeup) that BCE was being used. Since we are doing balancing BCE should work better.

Mixover. Inspired by mixup, mixover takes a bunch of (unbalanced) TP and FPs (which strictly speaking are TNs) and creates combinations of them so that the resulting labels can be supervised (1+NaN=1, 0+NaN=NaN,0+0=0) making sure linear interpolation is not destructive (beta,alpha=4; clip to 0.2,0.8); then it samples from the resulted mixed items computing class distribution so that resulting samples are balanced. Code is a bit tricky, but still:

```
class MixOver(MixHandler):
    "Inspired by implementation of https://arxiv.org/abs/1710.09412"
    def __init__(self, alpha=): super().__init__(alpha)
    def before_batch(self):
        ny_dims,nx_dims = len(self.y.size()),len(self.x.size())
        bs=find_bs(self.xb)
        all_combinations = L(itertools.combinations(range(find_bs(self.xb)), 2))
        lam = self.distrib.sample((len(all_combinations),)).squeeze().to(self.x.device).clip(0.2,0.8)
        lam = torch.stack([lam, 1-lam], 1)
        self.lam = lam.max(1)[0]
        comb = all_combinations
        yb0,yb1 = L(self.yb).itemgot(comb.itemgot(0))[0],L(self.yb).itemgot(comb.itemgot(1))[0]
        yb_one = torch.full_like(yb0,np.nan)
        yb_one[yb0>0.5] = yb0[yb0>0.5]
        yb_one[yb1>0.5] = yb1[yb1>0.5]
        yb_two = torch.clip(yb0+yb1,0,1.)
        yb_com = yb_one.clone()
```

```
        yb_com[~torch.isnan(yb_two)] = yb_two[~torch.isnan(yb_tw
o)]
        n_ones_or_zeros=(~torch.isnan(yb_com)).sum()
        ones=torch.sum(yb_com>=0.5,dim=1)
        zeros=torch.sum(yb_com<0.5,dim=1)
        p_ones = (n_ones_or_zeros/(2*( ones.sum())))/ones
        p_zeros= (n_ones_or_zeros/(2*(zeros.sum())))/zeros
        p_zeros[torch.isinf(p_zeros)],p_ones[torch.isinf(p_ones)
]=0,0
        p=(p_ones+p_zeros).cpu().numpy()/(p_ones+p_zeros).sum().
item()
        shuffle=torch.from_numpy(np.random.choice(yb_com.size(0)
,size=bs,replace=True,p=p)).to(self.x.device)
        comb = all_combinations[shuffle]
        xb0,xb1 = tuple(L(self.xb).itemgot(comb.itemgot(0))),tup
le(L(self.xb).itemgot(comb.itemgot(1)))
        self.learn.xb = tuple(L(xb0,xb1).map_zip(torch.lerp,weig
ht=unsqueeze(self.lam[shuffle], n=nx_dims-1)))
        self.learn.yb = (yb_com[shuffle],)
```

Augmentations. Time jitter (10% of time length), white noise (3 dB).

External data. We used Xeno Canto A-M and N-Z recording and sinc e they have 264 species we made the wild assumption that if we h ave 24 species and they have 10X randomly sampling would give yo u a "right" label (TN) 90% of the time¿ the goal was to add jung le/rainforest diversity at the expense of a small noisy labels w hich were accounted for labeling these weak labels as 0.1 (vs 0) .

Pseudolabeling. We also pseudolabeled using OOF models the non l abeled parts of training data to mine more TPs, and manually bal anced the resulting pseudo-label TPs.

Code. Pytorch, Fastai.

Final thoughts. It was a very fun competition and I am very glad of achieving a gold medal with a very short time, I want to tha nk Kaggle, sponsor and competitors for the competition and discu ssions; and of course my teammates @jpison @amezet @pavelgonchar for inviting me giving me the small nudge I needed to join ¿

Edit: I've added no hand labels to the title because this compet ition was VERY unique in that Kaggle effectively allowed hand la beling and that's very unusual. (Re: hand labeling vs external d ata, it was also very uncommon not having to disclose which exte rnal data you used during competition).

==> top6_b.txt <==
After @antorsae post (https://www.kaggle.com/c/rfcx-species-audi o-detection/discussion/220446) with his perspective, I'd like to give more information that could be useful to understand the in fluence of the differents experiments we did with our Tropic Mod el.

First, find here our journey, under submissions perspective:
Submissions

@jpison and @amezet join in a team on January 19th
@pavelgonchar joined the team on February 1st
@antorsae joined the team on February 10th

We worked in two different models (without hand labelling):

    SED model. We worked with this base model, improving with mixup, intelligent cropping, better schedule: https://www.kaggle.com/gopidurgaprasad/rfcx-sed-model-stater.
    We'd like to thank @gopidurgaprasad
    We reached with this model 0.91760 in Public (0.92270 in Private)
    Tropic Model
    @antorsae explained in his post the details of the models.

We learnt that the more diversity we have, the better score we got. So we did the experiments changing a lot the barebones, taking different from the complete list of TIMM models: https://github.com/rwightman/pytorch-image-models/blob/master/results/results-imagenet.csv

To select our two submissions, we prepare two scheme of ensembling:

Submission selected 1:
Scheme 1

Submission selected 2:
Scheme 2

We could did experiments very fast because we have a lot of computing power. For this competition we used the following GPUs:
@antorsae
6x3090
@amezet
3x3090 + 2x2080Ti
@jpison
1x3090
@pavelgonchar
2x3090

==> top7.txt <==
Thanks, Kaggle and RFCx, for this audio competition, and special thanks to my teammate @gaborfodor Without him, I probably would have given up a long time ago, somewhere at 0.8xx.
Data preparation

Resampling everything to 32kHz and split the audio files into 3 seconds duration chunks. We used a sliding window with a 1-second step.
Collecting more label

The key to our result was that Beluga collected tons of training samples manually. He created an awesome annotation application; you can find the details here. Source code included.
After the first batch of manually labeled examples, we quickly achieved 0.93x with an ensemble of a varying number of PANN (cnn14) models.

Input

We used mel-spectrograms as inputs with various n_bin (128, 192, 256, 288). Beluga trained PANN-cnn14 models with one input channel. For the other backbones (effnets, resnets, etc) I used three input channels with a simple trick:

I used different n_mel and n_fft settings for every channel. E.g. n_mel=(128, 192, 256), n_fft=(1024, 1594, 2048). This results in different height images, so resizing to the same value is necessary.
We both used torchlibrosa to generate the mel-spectrograms.

Augmentation

We used three simple augmentations with different probability:
Roll

```
np.roll(y, shift=np.random.randint(0, len(y)))
```

Audio mixup

```
w = np.random.uniform(0.3, 0.7)
mixed = (audio_chunk + rnd_audio_chunk * w) / (1 + w)
label = (label + rnd_label).clip(0, 1)
```

Spec augment

```
SpecAugmentation(time_drop_width=16, time_stripes_num=2, freq_drop_width=16, freq_stripes_num=2)
```

Architectures

- PANN - cnn14
- EfficientNet B0, B1, B2
- Densenet 121
- Resnet-50
- Resnest-50
- Mobilnet v3 large 100

We trained many versions of these models with different augmentation settings and training data. Beluga used a PANN - cnn14 model (I think it is the same as the original) from his Cornell solution.
I trained a very similar architecture with different backbones and I used attention head from SED:

```
x = ...generate mel-specotrogram...
x = self.backbone.forward_features(x)
x = torch.mean(x, dim=2)
x1 = F.max_pool1d(x, kernel_size=3, stride=1, padding=1)
x2 = F.avg_pool1d(x, kernel_size=3, stride=1, padding=1)
x = x1 + x2
x = F.dropout(x, p=0.5, training=self.training)
x = x.transpose(1, 2)
x = F.relu_(self.fc1(x))
x = x.transpose(1, 2)
x = F.dropout(x, p=0.5, training=self.training)
(clipwise_output, norm_att, segmentwise_output) = self.att_block
```

```
(x)
segmentwise_output = segmentwise_output.transpose(1, 2)
framewise_output = interpolate(segmentwise_output, self.interpol
ate_ratio)
output_dict = {
   "framewise_output": framewise_output,
   "clipwise_output": clipwise_output,
}
```

Training

Nothing special. Our training method was the same for all of the models:

- 4 folds
- 10 epochs (15 with a higher probability of mixup)
- Adam (1e-3; *0.9 after 5 epochs)
- BCE loss (PANN version)
- We used the weights of the best validation LWRAP epoch for inference.

Pseudo labeling

After we had an excellent ensembled score on the public LB (0.950), we started to add pseudo labels to our dataset. The result after we re-trained everything was 0.96x.

Final Ensemble

Our final ensemble had 80+ models (all of them trained with 4-folds)=== q ===
Thanks for the writeup, very strong PANN models and augmentation

==> top8.txt <==
Thank you for opening this competition
Also the notebooks and discussions have helped me a lot, thank you all!

My solution was similar to Beluga & Peter in 7th Place.
@https://www.kaggle.com/c/rfcx-species-audio-detection/discussion/220443.

- Multi-class multi-label problem
- Data cleaning train_data with hand labels
- SED model

In my case, Pseudo labeling did not work well, so I did not use it.

[hand labels]
While observing the data from t_min and t_max in the given train__tp.csv, I found that there are many kinds of bird calls mixed together.
So I decided to treat it as a multi-class multi-label problem.
It was also mentioned in the discussion that the test labels were eventually carefully labeled by humans.
The TPs in the given t_min, t_max range are all easy to understand, but there are many TPs in the 60s clip that are difficult to understand and not labeled.

I thought it would be better to label them carefully by myself t
o make the condition as close to test as possible in case such i
ncomprehensible calls are also labeled in test.
And I was thinking of doing Pseudo labeling after the accuracy o
f the model improves.

I trimmed the 5s~ around t_min and t_max in train__tp.csv.
Hand labels took about a week.
As a result, a total of 2428 clips and 5s chunks were used as tr
ain_data.
The distribution of the train_data classes looks like this
(I couldn't upload the image, so I'll post it later)

class nb
s3 1257
s12 520
s18 512
:
s16 100
s17 100
s6 97

I can see that there is a label imbalance, especially for s3, s1
2, and s18, because their labels co-occur among the other classe
s of clips.

In particular, s3 is dominant, so it tends to output high probab
ility, while a few classes output low probability, so i thought
this is a bad problem for this evaluation index.
Therefore, in order to achieve a more balanced distribution, I o
versampled the minority classes and undersampled the majority cl
asses.
However, the LB became worse.
Looking back, I didn't think of approaching the test distributio
n, as Chris pointed out.
https://www.kaggle.com/c/rfcx-species-audio-detection/discussion
/220389

I finally stopped doing class balancing and trust the LB and ens
embled 15models.

[trainning]
example single model
PANNsDense161 (public_LB 0.95548, private_LB 0.96300)

I also tried EfficientNet_b0, Dense121, etc., but Dense161 worke
d well.

train_data(sr=48000,5s)
window_size=2048,hop_size=512,mel_bins=256
MultilabelStratifiedKFold 5fold
BCEFocalLoss(¿=0.25,¿=2)
GradualWarmupScheduler,CosineAnnealingLR(lr = 0.001,multiplier=1
0,epo35)

Augmentation
GaussianNoise(p=0.5)
GaussianSNR(p=0.5)

FrequencyMask(min_frequency_band=0.0, max_frequency_band=0.2, p=0.3)
TimeMask(min_band_part=0.0, max_band_part=0.2, p=0.8)
PitchShift(min_semitones=-0.5, max_semitones=0.5, p=0.1)
Shift(p=0.1)
Gain(p=0.2)

[inference]
stride=1
framewise_output max
No TTA (I used it in the final ensemble model)

Finally, I've uploaded the train_data_wav (sr=48000) and csv that I used.
https://www.kaggle.com/shinoda18/rainforest-data

==> top9.txt <==
Thanks Kaggle and RFCx for a fun competition. My final submission without post process achieves private LB 0.926 and with post process achieves private LB 0.963! That's +0.037 with post process!
How To Score LB 0.950+

The metric in this competition is different than other competitions. We are asked to provide submission.csv where each row is a test sample, and each column has a species prediction.

In other competitions, the metric computes column wise AUC. In this competition, the metric is essentially row wise AUC. Therefore we need each column to represent probabilities. The columns of common species need to be large values and the columns of rare species need to be small values.
Train Distribution

The distribution of the train data has roughly 6x the number of false positives versus true positives for each species. If you train your model with that data, then when your model is unsure about a prediction, it will predict the mean that it observed in the train data which is 1/7. Therefore if it is unsure about species 3, it will predict 1/7 and if it is unsure about species 19, it will predict 1/7.

This is a problem because species 3 appears in roughly 90% of test samples whereas species 19 appears in roughly 1% of test samples. Therefore when your model is unsure , it should predict 90% for species 3 and 1% for species 19.
Test Distribution - Post Process

In order to correct our model's predictions we scale the odds. Note that scaling odds doesn't affect predictions of 0 and 1. It only affects the unsure middle predictions.

First we convert the submission.csv column of probabilities into odds with the formula
odds=p1¿p

Then we scale the odds with
new odds=factor¿old odds

And lastly we convert back to probabilities
prob=new odds1+new odds
Sample Code

```
# CONVERT PROB TO ODDS, APPLY MULTIPLIER, CONVERT BACK TO PROB
def scale(probs, factor):
    probs = probs.copy()
    idx = np.where(probs!=1)[0]
    odds = factor * probs[idx] / (1-probs[idx])
    probs[idx] =  odds/(1+odds)
    return probs

for k in range(24):
    sub.iloc[:,1+k] = scale(sub.iloc[:,1+k].values, FACTORS[k])
```

Increase LB by +0.040!

The only detail remaining is how to calculate the FACTORS above. There are at least 3 ways.

Create a model with output layer sigmoid, not softmax. Train with BCE loss using both true and false positives. Predict the test data. Compute the mean of each column. Convert to odds and divide by the odds of training data.

Probe the LB with submission.csv of all zeros and one column of ones. Then use math to compute factor for that species. UPDATE use random numbers less than 1 instead of 0s to avoid sorting uncertainty.

Use the factors listed in RFCx's paper here, Table 2 in Section 2.4

Personally, i used the first option listed above. I didn't have 5 days to probe the LB 24 times for the second option. And I didn't find the paper until the last day of the competition for the third option.

My single model scores private LB 0.921 without post process and scores private LB 0.958 with post process. Ensembling a variety of image sizes and backbones increased the LBs to 0.926 and LB 0.963 respectively.
Model Details

I converted each audio file into Mel Spectrogram with Librosa feature.melspectrogram and power_to_db using sampling rate 32_000, n_mels = 384, n_fft=2048, hop_length=512, win_length=2048. This produced NumPy arrays of size (384, 3751). I later normalized them with img = (img+87)/128 and trained with random crops of 384 x384 which had frequency range 20Hz to 16_000Hz and time range 6.14 seconds. Each crop contained at least 75% of a true positive or false positive.

I concatenated the true postive and false positive CSV files from Kaggle. My dataloader provided labels, masks, and one color images. The labels and masks were contained in the vector y which was 48 zeros where 2 were potentially altered. For species k, the kth element was 0 or 1 corresponding to false positive or true positive respectively. And the k+24th element was 1 indicating

mask true to calculate loss for the kth species. I trained TF mo
del with the following loss

```
 def masked_loss(y_true, y_pred):

     mask = y_true[:,24:]
     y_true = y_true[:,:24]

     y_pred = tf.convert_to_tensor(y_pred)
     y_true = tf.cast(y_true, y_pred.dtype)
     mask = tf.cast(mask, y_pred.dtype)
     y_pred = tf.math.multiply(mask, y_pred, name=None)

     return K.mean( K.binary_crossentropy(y_true, y_pred), axis=
-1 )*24.0
```

I used EfficientNetB2 with albu.CoarseDropout and albu.RandomBri
ghtnessContrast. The optimizer was Adam with learning rate 1e-3
and reduce on plateau factor=0.3, patience=3. The final layer of
 the model was GlobalAveragePooling2D() and Dense(24, activation
='sigmoid'). I monitored val_loss to tune hyperameters.
Try PP on Your Sub

If you want to try post process on your submission.csv file, I p
osted a Kaggle notebook here. It uses the 3rd method above for c
omputing FACTORS. It also has 3 MODE you can try to account for
different ways that you may have used to train your models.

==> top10.txt <==
Thank you to the organisers, Kaggle, and to everyone who shared
ideas and code for this competition. I learned a lot, as I'm sur
e many of you have, and I thought I would break down my approach
 since I know many competitors couldn't find a way to use the Fa
lse Positive labels. I'm thrilled to have secured my first gold
(and solo gold) in a competition, so it's the least I could do!
Summary

On a high-level, my approach was as follows:

    train models using existing labels
    generate pseudo-labels on train/test
    isolate the frames which had a very high/low prob across an
ensemble of models
    conservatively threshold these and use as new TP/FP values f
or the relevant class
    repeat

This gradually increased the amount of data I had to work with,
until I had at least 2 frames from each recording with an identi
fied TP and/or FP. The growing data diversity allowed successive
 models to generalise better.
What Worked

Fixed Windows

I used a window of 5 seconds centred on the TP/FP. For predictin
g on test, I used overlapping windows with a step of 2.5 seconds
. This ensured that train/test had identical preprocessing for t

heir inputs. The maximum value was taken for each class across a
ll of these windows. For some submissions, I used the average of
 the top 3 predictions but this didn't seem to notably change th
e LB score.

Agnostic Loss

Perhaps a better term already exists for this, but this was what
 I called my method for using both the TPs and FPs in a semi-sup
ervised fashion. The problem we face with unlabelled data is tha
t any spectrogram can contain multiple classes, so setting the t
arget as 0 for everything apart from the given label will penali
se true positives for other classes. We can only know for sure t
hat one class is present or absent, and the loss needs to reflec
t this. So I excluded all non-definite targets from the loss cal
culation. In the target tensor, a TP is 1 while an FP is 0. Unla
belled classes are given as 0.5. These values are then excluded
 from the loss calculation. So if we had 5 classes (we have 24, b
ut I'm saving room here) and this time window contained a TP for
 class 0 and an FP for class 3:

```
y = torch.Tensor([1., 0.5, 0.5, 0., 0.5])
```

And in the loss calculation:

```
preds = model(inputs)
preds[targets==0.5] = 0
loss = BCEWithLogitsLoss(preds, targets)
loss.backward()
```

Thus the model is 'agnostic' to the majority of the potential la
bels. This allows the model to build a guided feature representa
tion of the different classes without being inadvertently given
 false negatives. This approach gave me substantially better LB s
cores.

The figure of 0.5 is arbitrary and could've been any value apart
 from 0 or 1: the salient point is that the loss resulting from
 unlabelled classes is always constant. Note that this kind of in
place operation is incompatible with nn.Sigmoid or its functiona
l equivalent when performing backprop so you need to use the raw
 logits via torch.nn.BCEWithLogitsLoss().

ResNeSt

I found EfficientNet to be surprisingly poor in this competition
, and all of my best scores came from using variants of ResNeSt
 (https://github.com/zhanghang1989/ResNeSt) paper available here.

For 3-channel input I used the librosa mel-spectrogram with powe
r 1, power 2 and the delta function to capture temporal informat
ion. With some models I experimented with a single power-1 spect
rogram, delta and delta-delta features instead. While quicker to
 preprocess, I noticed no impact on scores.

I also incorporated it into the SED architecture as the encoder.
 This showed very promising metrics during training, and while s
adly I didn't have time to run a fully-convergent example its in

clusion still helped my score. In future competitions this could be a very useful model. ResNeSt itself only takes a 3-channel input and has no inbuilt function to extract features, so I had to rejig it to work properly: I'll be uploading a script with that model shortly in case anyone is interested.

Augmentations

From Hidehisa Arai's excellent kernel here, I selected GaussianNoiseSNR(), PinkNoiseSNR(), TimeShift() and VolumeControl(). I was wary of augmentation methods that blank out time windows or frequency bands like SpecAugment. Some of the sounds occur in a very narrow frequency range (e.g. species 21) or in a very narrow time window (e.g. species 9) and I didn't want to make 'empty', positive samples that would coerce the model into learning spurious features. I also added some trivial augmentations of my own:

- swapping the first and last half of the audio vector
- adding a random constant before spectrogram normalisation (occluding the relevant features)
- 'jiggling' the time window around the centre of t_mid, at a maximum of 1 second offset in either direction

Eliminating species 19

A minor point, but this class was so rare that setting all of its predictions to zero usually improved the LB score by 0.001. There were many unrelated sounds that would presumably cause the model to produce false positives. My best submission didn't do this however; it was a simple blend of models including ResNeSt-50, ResNeSt-101, EfficientNet-b1 and the SED architecture described above. I used weighted averaging roughly in proportion to the individual models' performance.

What didn't work

- Using separate models for different frequency ranges (these models never gained an adequate feature representation, and produced many false positives).
- EfficientNet alone gave poor results, but helped as part of an ensemble.
- Larger models (ResNeSt101, EfficientNet b3) didn't improve scores.
- TP-only training.
- Models that worked on all windows for a single clip - these were slow and produced inferior results.

Otherwise I was quite lucky - I thought about my methodology for a while and most of what I tried worked well on the first attempt. If I'd had more time, I would have liked to try:

- automatically labelling some species as an FP of a similar class (e.g species 9 & 17)
- probing the LB for class distribution (I suspect you could get +0.9 by only predicting the most common half of the classes and ignoring everything else) - I realised the importance of this too close to the deadline
- experimenting with different encoders for the SED architecture.

using a smaller window size (<=3 seconds) for greater fideli
ty.

The overall class prediction histograms for my final submission
were as follows:

Some classes gave me particular trouble. I used my own, simple s
coring metric during training that recorded the proportion of po
sitive cases that were predicted above a certain threshold. I ne
ver satisfactorily made a model that could detect the rarer clas
ses like 6, 19 or 20 in a reliable fashion.

Overall I had an interesting time exploring how to work with the
 application of CNNs to spectrograms, and with large amounts of
unlabelled data. In the next audio competition, perhaps I'll aim
 a little higher! I'm looking forward to seeing how those who sc
ored > 0.97 managed to achieve their results.

If you have any questions I'll do my best to answer them!

==> top11.txt <==
First of all, thanks to the host, for providing yet another very
 interesting audio challenge. The fact that it was only partly l
abelled was a significant difference with previous bird song com
petition.

Second, congrats to all those who managed to pass the 0.95 bar o
n public LB. I couldn't, and, as I write this before competition
 deadline, I don't know why. I tried a lot of things, and it loo
ks like my approach has some fundamental limit.

Yet it was good enough to produce a first submission at 0.931, p
lacing directly at 3rd spot while the competition had started tw
o months earlier.

This looked great to me. In hindsight, if my first sub had been
weaker, then I would not have stick to its model and would have
explored other models probably, like SED or Transformers.

Anyway, no need to complain, I learned a lot of stuff along the
way, like how to efficiently implement teacher student training
of all sorts. I hope this knowledge will be useful in the future
.

Back to the topic, my approach was extremely simple: each row of
 train data, TP or FP, gives us a label for a crop in the (log m
el) spectrogram of the corresponding recording. If time is x axi
s and frequency the y axis, as is generally the case, then t_min
, t_max gives bounds on x axis, and f_min, f_max gives bounds on
 the y axis.

We then have a 26 multi label classification problem (24 species
 but two species have 2 song types. I treated each species/song
type as a different class). This is easily handled with BCE Loss
.

The only little caveat is that we are given 26 classes (species
+ song type) but we get only one class label, 0 or 1 per image.

We only have to mask the loss for other classes and that's it!

I didn't know it when I did it, but a similar way has been used by some of the host of the competition, in this paper (not the one shared in the forum): https://www.sciencedirect.com/science/article/abs/pii/S0003682X20304795

The other caveat is that the competition metric works with a label for every class. Which we don't have in train data. However the competition metric is very similar to a roc auc score per recording: when a pair of predictions is in the wrong order, i.e. a positive label has a prediction lower than another negative label prediction, then the metric is lowered. As a proxy I decided to use roc-auc on my multi label classification problem. Correlation with public LB is noisy, but it was good enough to let me make progress without submitting for a while.

What worked best for me was to not resize the crops. It means my model had to learn from sometimes tiny images. To make it work by batch I pad all images to 4 seconds on the x axis. Crops longer than that were resized on the x axis. Shorter ones were padded with 0. One thing that helped was to add a positional encoding on the frequency axis. Indeed, CNNs are good at learning translation independent representations, and here we don't want the model to be frequency independent. I simply added a linear gradient on the frequency axis to all my crops.

For the rest my model is exactly what I used and shared in the previous bird song competition: https://www.kaggle.com/c/birdsong-recognition/discussion/183219 Just using the code I shared there was almost good enough to get 0.931. The only differences are that I add noise as in the first solution in that competition. I also did not use a no call class here, nor secondary labels.

For prediction I predict on slidding crops of each test recording and take the overall max prediction. This is slow as I need to do each of the 26 classes separately. This is also maybe where I lost against others: my model cannot learn long range temporal patterns, nor class interactions.

With the above I entered high with an Efficient B0 model, and moved to 0.945 in few submissions with Efficientnet B3 . Then I got stuck for the remainder of the competition.

I was convinced that semi supervised learning was the key, and I implemented all sorts of methods, from Google (noisy student), Facebook, others (mean student). They all improved weaker models but could not improve my best models.

In the last days I looked for external data with the hope that it would make a difference. Curating all this and identifying which species correspond to the species_id we have took some time and I only submitted models trained with it today. They are in same range as previous ones unfortunately. with a bit more time I am sure it could improve score, but I doubt it would be significant..

For matching species to species id I used my best model and pred

icted the external data. It would be interesting to see if I got this mapping right. Here is what I converged to :

0 E. gryllus
1 Leutherodactylus brittoni
2 Leptodactylu albilabris
3 E. coqui
4 E. hedricki
5 Setophaga angelae
6 Melanerpes portoricensis
7 Coereba flaveola
8 E. locustus
9 Margarops fuscatus
10 Loxigilla portoricensis
11 Vireo altiloquus
12 E. portoricensis
13 Megascops nudipes
14 E. richmondi
15 Patagioenas squamosa
16 Eleutherodactylus antillensis
17 Turdus plumbeus
18 E. unicolor
19 Coccyzus vieilloti
20 Todus mexicanus
21 E wightmanae
22 Nesospingus speculiferus
23 Spindalis portoricensis

The picture in the paper shared in the forum helped to disambiguate few cases: https://reader.elsevier.com/reader/sd/pii/S1574954120300637 The paper also gives the list of species. My final selected subs did not include models trained on external data, given they were not improving.

This concludes my experience in this competition. I am looking forward to see how so many teams passed me during the competition. There is certainly a lot to be learned.

Edit. I am very pleased to get a solo gold in a deep learning competition., This is a first for me, and it was my goal here.

Edit 2: The models I trained last day with external data are actually better than the ones without. The best one has a private LB of 0.950 (5 folds). However, they are way better on private LB but not on public LB. Selecting them would have been an act of faith. And late submission show they are not good enough to change my rank. No regrets then.

Edit 3 Using Chris Deotte post processing. my best selected sub gets 0.7390 on private LB. It means that PP was what I missed and that my modeling approach was good enough probably. I'll definitely look at test prediction distribution from now on!
Quote
Follow
Report
90 Upvoters

Comments (62)

Sort by

Hotness
Insert Quote

CPMPTopic Author ¿ (11th in this Competition) ¿ 4 days ago ¿ Options ¿ Report ¿ Reply

2

I didn't think of post processing but I did think of pseudo labeling during the competition, yet could not make it work. After reading all the writeups where teams who passed me successfully used PL I revisited what i did. The issue was that instead of randomly sample crops for PL, I imposed a distribution that matches training samples, i.e. same number of positive pseudo labels per class. When I remove this bias then pseudo labeling works. In my first experiment, a single model gets a 0.01 boost on public and private LB. With tuning and iterations I now see how I could have moved higher.

I am not sure why I imposed this sampling bias. I'll try to be more careful next time.

ryches ¿ (26th in this Competition) ¿ 8 days ago ¿ Options ¿ Report ¿ Reply

4

Kind of kicking myself for not exploring this further. The first thing I sent to the team when I joined them was showing them a model I had that was kind of similar to yours. Instead of cropping though I simply masked out the regions outside of the frequency band and time that were irrelevant. So I would crop around the region of the time and then I would mask out the frequencies that were irrelevant to that prediction.

Would look something like this. I made it so they were long enough that I never had to do any cropping, only ever padding near the beginning or end of the audio. Would use similar procedure to you at test time, but I could stack all of them together fairly easily so it was 16(num_time_steps)*24(num_frequency_ranges), 128(num_mel_bins), 500 (time) and it was fairly fast to do inference. On the crops themselves I got validation performance of .985 at times, but when I applied the rolling window validation I would get poor results like .6-.7.

I tried the masked loss, but I did not apply it to this specific model and I never made a submission with it because the windowed validation looked poor. I considered the linear gradient to give position but ended up not using it in this setup because I figured the model already had its relative position based on the amount of 0 padding above and below the unmasked signal. Might have to fiddle with that and see if it was actually good.

CPMPTopic Author ¿ (11th in this Competition) ¿ 8 days ago ¿ Opt

ions ¿ Report ¿ Reply

1

Instead of cropping though I simply masked out the regions outside of the frequency band and time that were irrelevant. So I would crop around the region of the time and then I would mask out the frequencies that were irrelevant to that prediction.

That's what I did. Crop then pad. Sorry if this is not clear enough in my post.

It looks like you had the same idea as me ;)

==> top11_link_cornell.txt <==
First of all I want to thank the host and Kaggle for this very challenging competition, with a truly hidden test set. A bit too hidden maybe, but thanks to the community latecomers like us could get a submission template up and running in a couple of days. I also want to thank my team mate Kazuki, without whom I would probably have given up after many failed attempts to beat the public notebook baseline¿

Overview

Our best subs are single efficientnet models trained on log mel spectrograms. For our baseline I started from scratch rather than reusing the excellent baselines that were available. Reason is that I enter Kaggle competition to learn, and I learn more when I try from scratch than when I modify someone else' code. We then evolved that baseline as we could in the two weeks we had before the end of competition.

Given the overall approach is well known and probably used by most participants here I will only discuss the items that may be a bit different from what others did.

Training data clips

It was clear from host that training on random 5 second clips had a drawback: some of the clips may not contain the target bird. We then used a simple hypothesis: clips were stripped, i.e. periods without a song at the beginning or at the end were removed for the sake of reducing storage needs. We therefore trained on first 5 seconds or last 5 seconds of clips, assuming these would contain the target bird. We preprocessed all data to be sampled at 32 kHz.

Noise
We added noise extracted from the two test sequences made available, a bit like what Theo Viel did. But we used the meta data to extract sub sequences without bird call, then we merged these sequence with a smooth transition between them. We then added a random clip of the merges sequences to our training clips

No Call
We added the freefield1010 clips that were labelled as nocall to our training data. We added a 265th class to represent the no c

all. As a result our model could predict both one or more birds, and a nocall. Adding this data and the nocall class was probably the most important single improvement we saw in CvV and LB scores. It is what led us to pass the public notebook baseline.

Multi Bird Clips
The main documented difference between train and test data is that train is a multi class data while test is a multi label data. Therefore we implemented a mixup variant were up to 3 clips could be merged. This is not really mixup as the target for the merged clip is the maximum of the targets of each merged clip.

Secondary labels
Primary labels were noisy, but secondary labels were even noisier. As a result we masked the loss for secondary labels as we didn't want to force the model to learn a presence or an absence when we don't know. We therefore defined a secondary mask that nullifies the BCE loss for secondary labels. For instance, assuming only 3 ebird_code b0, b1, and b2, and a clip with primary label b0 and secondary label b1, then these two target values are possible:

[1, 0, 0]

[1, 1, 0]

The secondary mask is therefore:

[1, 0, 1]

For merged clips, a target is masked if it it not one of the primary labels and if it is one of the secondary labels.

Loss Function

We use binary cross entropy on one hot encoding of ebird codes. Using bce rather than softmax makes sense as bce extends to multi label seamlessly. We tried dice loss to directly optimize F! score, but for some reason this led to very strong overfitting.

Class Weights

The number of record per species is not always 100. In order to not penalize the less frequent one we use class weights inversely proportional to class frequencies. And for the nocall class we set it to 1 even though it was way more frequent than each bird classes to make sure the model learns about nocall correctly.

Model

Our best model was efficientnet on log mel spectrograms. We resized images to be twice the size of effnet images: 240x480 for effnet b1, 260x520 for effnet b2, and 300x600 for effnet b3. We started from efficientnet_pytorch pretrained models. We tried the attention head from PANNs models but it led to severe overfitting. I am not sure why to be honest, maybe we did something wrong.

Training

Nothings fancy, adam optimizer and cosine scheduler with 60 epoc hs. In general last epoch was the one with best score and we use d last epoch weights for scoring.

Log Mel Spectrogram

Nothing fancy, except that we saw a lot of power in low frequenc ies of the first spectrograms we created. As a result we clipped frequency to be at least 300 Hz. We also clipped them to be bel ow 16 kHz given test data was sampled at twice that frequency.

Augmentations

Time and pitch variations were implements in a very simple way: modify the length of the clipped sequence, and modify the sampli ng rate, without modifying the data itself. For instance, we wou ld read 5.2 seconds of a clip instead of 5 seconds, and we could tell librosa that the sampling rate was 0.9*32 kHz. We then com pute the hop so that the number of stft is equal to the image wi dth for the effnet model we are training. We also computed the n umber of mel bins to be equal to the height of the image. As a r esult we never had to resample data nor resize images, which spe ed up training and inferencing quite a bit. There was an issue w ith that still: this led us to use a high nftt value of 2048 whi ch lead to poor time resolution. We ended up with nfft of 1048 a nd a resize of the images in the height dimension.

Cross Validation

We started with cross validating on our training data (first or last 5 seconds of clips) but it was rapidly clear that it was no t representative of the LB score. And given we had very few subm issions, we could not perform LB probing. We therefore spent som e time during last week to create a CV score that was correlated with the public LB. Our CV score is computed using multi bird c lips for 46% of the score, and nocall clips for 54% of the score . We added warblrb and birdvox nocall clips to the freefield1010 for valuating no call performance. We tuned the proportion of e ach possible number of clips in multi bird clips, and the amount of noise until we could find a relatively good CV LB relationsh ip, see the picture below: x is cv, y is public lb.

The correlation with private LB is also quite good:

We then used it to guide our last few submissions training and t hresholding. Our CV said 0.7 was best, and late submissions prov ed it was right. We ended up selecting our 2 best CV submissions , which were also the 2 best public LB submissions, and also the two best private Lb submissions.

Conclusion

These were probably the most intensive two weeks I had on Kaggle since a long time. My first two subs 14 days ago were a failure , then a LB score of 0. Kazuki had started a bit earlier, but no t much. I am very happy about where we landed, and I am not sure we would have done much better with few more days. Maybe using effnet b4 or b5 wold have moved us higher but I am not sure. I a

m looking for gold medalist solutions to see what we missed. I'm sure I'll learn quite a bit.

PS

You can see some of the above in the scoring notebook for our best sub: https://www.kaggle.com/cpmpml/infer-model-210

==> top13.txt <==
Hey Everybody, I wanted to dump my solution real quick in case anyone was interested.

It seemed to me that the critical issue is that there are a TON of missing labels. The provided positive examples data (train_tp.csv) has ~1.2k labels. The lb probing that @cpmpml did suggest 4-5 labels per clip on average. If the train data follows the same distribution we should expect ~21k labels, and that's just at the clip level. We'd expect to see multiple calls from the same bird per clip, i.e. multiple frame labels per clip label. My best models seemed to think there were closer to 40k labels.

So my idea was to do something along the lines of Noisy Student, where the general idea is to do progressive pseudo labeling where each successive model is larger and there's more noise applied to the training data. On its own, Noisy Student doesn't work very well, so I used a few other tricks.
1. Mean Teacher

My first setup looks super similar to what's going on in Mean Teachers Find More Birds. I train on a combo of centered positive examples and random unlabeled samples using consistency and BCE loss. Here, I'm using SED + resnet34 and some light augmentation: gaussian noise, frame/frequency dropout. This gets me to 0.865 on the public lb.

Using 5-fold mean-teacher models, I do OOF prediction to get pseudo labels over the entire training dataset.
2. Co-Teaching

Now I want to train on my pseudo labels, but it's safe to assume they're pretty noisy. To deal with the bias introduced by my new, noisy labels, I do something along the lines of Co-Teaching. Briefly, the idea is to train 2 models simultaneously on the same data, but with different augmentations applied to each. Then the samples with the highest loss from Model A are ignored when doing backprop in Model B and vice versa. The % of ignored samples gets ramped up slowly. The theory is that the models will learn the correct labels early in training and start to overfit to noise later on. By dropping potentially noisy labels, we avoid introducing a bad bias from our pseudo labels.

I modified the authors idea slightly for the competition. In my setup, it's impossible for either model to ignore the good labels from train_tp or train_fp. Only pseudo labels can be ignored. I believe this helps with class imbalance issues.

Using this setup with more aggressive augmentation and densenet121, I'm able to get to 0.906 on the public lb.

3. Heavy Mixup

Finally, using my second round of pseudo labels, I train on randomly sampled segments from all the training data. Here I apply even more aggressive augmentations and add mixup 60% of the time with a mixing weight sampled from Beta(5,5) (typically around 0.5). For mixup, any label present in either clip gets set to 1.0. I run this for 80 epochs. The prev 2 models were run for around 32 epochs. A 5 fold ensemble with this setup using densenet 121 gets me up to 0.940 on the public lb.

I¿m able to get to 0.943 by ensembling ~90 models taking the geometric mean.
Other Tricks

Centering the labels from train_tp in the sampled clip segment early on seemed to help.
When making predictions I¿m averaging 4 metrics: average and max clip-wise and frame-wise predictions.
Mixup only worked for me when it was done on the log mel spectrograms. Doing it on the audio didn't work.
Augmentations (intensities varied) (excluding mixup):

```
augmenter = A.Compose([
    A.AddGaussianNoise(p=0.5, max_amplitude=0.033),
    A.AddGaussianSNR(p=0.5),
    A.FrequencyMask(min_frequency_band=0.01, max_frequency_band=0.5, p=0.5),
    A.TimeMask(min_band_part=0.01, max_band_part=0.5, p=0.5),
    A.Gain(p=0.5)
])
```

Let me know if you have any questions!

==> top14.txt <==
14th Place Solution - Binary Classification on cropped frequency x time
Posted in rfcx-species-audio-detection 7 days ago

16

First of all thanks to Kaggle and Hosts for organizing this competition.

We (me & @ks2019) were initially working in Cassava Leaf Disease Classification but thanks to this post by @cpmpml that gave us the direction that something different needs to be tried then what is going on in the public. On closer inspection, we found something similar to him. The frequency-time crops in the audios for a specie_id are almost constant (i.e. say for specie_id 23 - in most of the recordings audio frequency lies between 6459 and 11628, and its duration lasts for about 16 seconds). This gave us the idea of cropping out all the potential regions from the spectrogram and perform binary classification on them.

Our approach can be summarized as -

Crop images from spectrogram with frequency ranging between

max-min frequency observed for a specie_id and with time duratio
n 2 times the max duration observed for a specie_id
    Pre-Process: resize crops to size 128 x 256, scale between 0
 and 1, and perform augmentation
    Train B0 binary classifier detecting the presence of specie
(a single binary classifier - here we tracked back using the fre
quency information of the crop that which ID we are asking class
ifier to detect for)
    Generate Pseudo-labels
    Retrain
    Perform inference on the test, and take the mean of max n(in
 our case it was 3) probabilities observed for a specie_id in a
recording as the probability of that specie_id

Note: On the very first submission, a single model with the abov
e approach gave us 0.921 as public LB (has private LB 0.927), th
en pseudo labeling and a little bit of blending took private LB
to 0.948
Cropping

From each spectrogram, for each specie_id x songtype, we cropped
 out image sequences with the frequency range between min and ma
x frequency observed for that specie_id x songtype, and then cre
ated image sequences with duration 2 times the max time interval
 the audio lasted for that specie_id x songtype in the train.
Img
Here - In case img not visible
Augmentation

Along with adding random noise, we took a false positive sample
of the same specie_id and added that to the audio sample. After
this augmentation, the label of the recording id x specie id rem
ained the same(i.e. a false-negative remained the false negative
 and a true positive remained the true positive)

==> top17.txt <==
Congratulations to all the winners and gold getters, I guess tho
se teams that broke the 0.950 wall have found the essence of thi
s competition, which we couldn't.

Firstly, thanks to the host for holding quite an interesting com
petition. Partly labeled classification is a challenging task, w
hich made this competition more interesting than a simple bioaco
ustics audio tagging competition.

Our solution is a ranking average of image classification models
 and SED models. Y.Nakama, kaerururu, and I worked a lot on SED
models but couldn't break 0.90 until we merge with Taku Hiraiwa.
 His model was based on image classification similar to @cpmpml'
s model. We found that quite good , so we focused on improving i
mage classification models in the remained days.
Image classification models

It was Taku Hiraiwa's idea to only use the annotated part of the
 train data. To do so, we crop the image patches from log-melspe
ctrogram of train data based on the t_min, t_max, f_min, f_max i

nformation of train_tp.csv and train_fp.csv, and resized the pat
ch to fixed shape (say, 320 x 320 or so). The cropping is perfor
med on the fly through training, so the part we crop out is rand
omized along with time axis.
With these image patches we trained EfficientNet models, and mon
itored F1 score with threshold 0.5.
Here's the other details of image classification models.

    image size: varies from (244, 244) to (456, 456) between mod
els
    backbone: EfficientNetB0 - B5 (used timm and used tf_efficie
ntnet_b<0-5>_ns weights).
    augmentation: GaussianNoise, Gain, PitchShift of audiomentat
ions on raw waveform. Also HorizontalFlip also had positive impa
ct on LB slightly, so we used (but don't know why it worked).
    AdamW optimizer with linear warmup scheduler
    BCEFocalLoss

    In the end, we trained a stacking model that takes the outpu
t of models below which achieve public 0.942:

    tf_efficientnet_b0_ns image size 244
    tf_efficientnet_b0_ns image size 320
    tf_efficientnet_b0_ns image size 456
    tf_efficientnet_b1_ns image size 456
    tf_efficientnet_b2_ns image size 456
    tf_efficientnet_b3_ns image size 456
    tf_efficientnet_b4_ns image size 456
    tf_efficientnet_b5_ns image size 456

SED models

All of our SED models use the head architecture introduced in PA
NNs repository. The CNN encoder is either EfficientNet or ResNeS
t, and they are trained with weak/strong supervision. We tried a
 lot of things on this, but couldn't find factors that consisten
tly work well - which means the LB scores varied quite randomly
w.r.t CV score.
Our best SED model is rank average of 11 models(public: 0.901, p
rivte: 0.911) below - each of them differs slightly so we descri
be the difference briefly.
2 x kaerururu's model (public: 0.882, 0.873)

    Based on public starter SED (EffnB0) notebook (https://www.k
aggle.com/gopidurgaprasad/rfcx-sed-model-stater)
    3ch input
    10sec clip
    waveform mixup
    some augmentation (audiomentations)
    pseudo-labeled datasets (add labels on tp data)
    trained with tp and fp dataset (1st training)
    trained with pseudo-labeled tp (2nd training)
    tta=10

5 x arai's model (public: 0.879,0.880, 0.868, 0.874, 0.870)

    Based on Birdcall's challenge 6th place (https://github.com/

koukyo1994/kaggle-birdcall-6th-place)
    ResNeSt50 encoder or EfficientNetB3 encoder
    AddPinkNoiseSNR / VolumeControl / PitchShift from My Noteboo
k
    tp only

4 x Y.Nakama's model (public: 0.871, 0.863, 0.866, 0.870)

    Based on Birdcall's challenge 6th place (https://github.com/
koukyo1994/kaggle-birdcall-6th-place)
    ResNeSt50 encoder or ResNet200D encoder
    mixup & some augmentations
    2nd stage training
        1st stage: weighted loss for framewise_logit & logit
        2nd stage: loss for logit
    tp only

==> top19.txt <==
[completed] rank 19th solution in 13 days - LB 0.937/0.939 (publ
ic/private) 5 fold model
Posted in rfcx-species-audio-detection 9 days ago

46

[summary]

    Design a sliding window conv classifier net. Trained with tp
 and fp annotations (no pesudo label) on resnet34, this architec
hiture achieved LB 0.937/0.939 (public/private) in 5 fold model.
 A single fold model gives 0.881/0.888 (public/private). The fir
st max pooling resnet34 is removed to make the stride of the bac
kbone 16 (instead of 32)

    Final submission is LB 0.945/0.943 (public rank 15/private r
ank 19) is an ensemble of this network with different CNN backbo
ne.

I am grateful for Kaggle and the host Rainforest Connection (RFC
x) for organizing the competition.

As a Z by HP & NVidia global datascience ambassador under https:
//datascience.hp.com/us/en.html.HP & Nvidia has kindly provided
 a Z8 datascience workstation for my use in this competition. Wit
hout this powerful workstation, it would not be possible for me
to develop my idea from scratch within 13 days.

Because I can finish experiments at a very great speed (z8 has a
x quadro rtx 8000, NVlink 96GB), I gain a lot of insights into m
odel training and model design when the experimental feedback is
 almost instantaneous. I want to share these insights with the k
aggle community.

Hence I started another thread to guide and supervise kagglers w
ho wants to improve their training skills. This is targeted to b
ring them from sliver to gold levels. You can refer to:
https://www.kaggle.com/c/rfcx-species-audio-detection/discussion
/220379

https://www.kaggle.com/c/hpa-single-cell-image-classification/discussion/217238

data preprocessing

log mel spectrogram with modified PCEN denoising (per- channel energy normalization). A 10sec clip has melspec of size 128x938
n_fft = 2048
win_length = 2048
hop_length = 512
num_freq = 128

augmentation

I haven't tried much augmentation yet. For TP annotation, random shift by 0.02 sec. For FP annotation, random shift width of the annotation and then random flip, mixup, cut and paste of the original annotation and its shift version.
Augmentation is done in the temporal time domain because of my code. (this is not the best solution)
It can be observed that if I increased the FP augmentation, the bce loss of the validation FP decreases and the public LB score improved.
heavy dropout in model

model and loss

please see the below images
during training, I monitor the separate log loss of validation TP and FP. I also use the LRAP assumpting one label (i.e. top-1 correctness). This is to decide when to early stop or decrease learning rate. These are not the best approach.

daily progress:

I enter the competition are reading CPMP post, saying that he has 0.930 LB in the first submission. WOW! Reading between the lines, it means:

the public kernels are doing it "wrong". It could be the problem setting, the network architecture, the data, or some magic features. there is something very fundamental that can do better than the public kernel.
0.930 is quite high score (in the gold region). With 2 more week to the deadline, I decided to give it a try. if I solved this unobvious puzzle, i may end up in the gold region as well.
the first step is to go through the public kernel (some PANN SED) and the baseline image classifier. I want to see what most people are doing (and could be improved)
after working for a week, I realize that there are two major "potential flaws":
treating it as a multi-instance learning SED. Well, MIL could be a solution, but the problem is that we don't have bag label (clip label). Most MIL required bag level label, but lack i

nstance level label (segment label).Here we have the opposite.
        not using FP in train. Most public kernel use only TP as train only.
    hence I start to design my own network architecture. The first step is an efficient way to do crop classification based on mean annotation box. Hence I design the slide window network.
    The next step is to use TP+FP in training
    The last step is to use pesudo labels, but I don't have time to complete this. But I do have some initial experimental results on this. Top-1 (max over time) pesudo labels is about 95% accurate for LB score of 0.94. This is good enough for distillation.
    Because pesudo labeling requires many probing to prevent error propagation, I could not do it because I don't have sufficient slots in the last days. Worst still, there is no train data at clip level at all. This makes local testing impossible.

I was able to train 5-fold efficientB0 and resnet50 in the last day. Because of the large GPU card of the HP workstation I am using, I can train large models with large batch size. When I compared training the same model with smaller batch size on my old machines, I find that the results are different and inferior, even if I use gradient accumulation.

I strongly feel that we have reached a new era. I think this is also why the latest RTX3090 has larger GPU memory than the previous cards. The future is the transformers ¿. meaning more GPU memory. That's is how fast deep learning are moving!

==> top21.txt <==
21st place solution - FP co-teaching with loss improvements
Posted in rfcx-species-audio-detection 9 days ago

19

Congratulations to the top finishers!

The whole my solution is here: https://github.com/MPGek/mpgek-rfcx-species-audio-detection-public.
I left the best configs that were used in the final submission.
Summary

Summary of my solution:

    General augmentations with frequency band augmentations:
        Frequency band filtering
        Frequency band mixup
    TP training with a combined loss of the BCE for confident samples and LSoft 0.7 for noisy samples
    FP training with a BCE for confident samples and ignored losses for samples with noisy samples
    FP co-teaching training with loss as described in the co-teaching paper, but with the extra loss for the high loss samples
    Ensemble of TP, FP, FP co-teaching results.

Spectrogram and random crop

For the training, I used a random crop with the centered sample

in 10 seconds and a random crop for 6 seconds.
For validation and prediction, I used 6 seconds crop with a stride in 2 seconds with maximizing outputs.
For the mel spectrogram, I used the following parameters:

- mels count: 380
- FTT: 4096
- window length: 1536
- hop length: 400
- fmin: 50
- fmax: 15000

So one sample in 6 seconds produced an image with size 720 x 380 pixels

Augmentations

Augmentation that improves LB:

- Gaussian noise
- Random crop + resize with size reduction in 40 and 20 pixels
- Frequency band filtering - based on f_min and f_max of the sample I set 0 values to all mels that lower or higher than f_min and f_max with a sigmoid transition to remove sharp edges
- Frequency band mixup - for some samples I used frequency band filtering and then I mixed it with different samples with a band that higher than f_max and with a band that lower than f_min. So I managed to get a single image with the 3 mixed samples.

Example of the Frequency band filtering (top - original sample, bottom - sample after filtering):

Example of the Frequency band mixup (top - original sample, bottom - sample after mixup):

Augmentation that doesn't improve LB:

- SpecAugment
- Sum mixup

Network topology

I got the best results on EfficientNetB2, B4, B7 (noisy students weights) with a simple FC to 24 classes after the adaptive average pool.

I tried to use different head but all of them gave the same or worse result:

- the hyper column
- the hyper column with auxiliary losses after each pooling
- extra spatial and channel attentions blocks - CBAM
- dense convolutions

TP training

Based on the post where described that every sound file can present unlabeled samples I have to work noisy samples.

I split all samples into confident samples and noisy samples:

confident samples - all sigmoid outputs for classes which present in the train_tp.csv file (1 in targets tensor)
noisy samples - all sigmoid outputs for classes which not described in the train_tp.csv file (0 in target tensor)

For the confident samples, I used simple BCE. For the noisy samples, I used LSoft with beta 0.7 (https://arxiv.org/pdf/1901.01189.pdf).
LSoft:

```
def forward(self, input: torch.Tensor, target: torch.Tensor):
        with torch.no_grad():
            pred = torch.sigmoid(input)
            target_update = self.beta * target + (1 - self.beta) * pred
        loss = F.binary_cross_entropy_with_logits(input, target_update, reduction=self.reduction)
        return loss
```

In the loss function, I flatten all outputs (even batch dim) to a linear array. And split items into 2 arrays where targets were 1 and where targets were 0.

With LSoft I got on EfficientNetB7 0.912-0.915 LB.
Without LSoft - BCE for all samples I got only about 0.895-0.900.
FP training

For the FP training, I used a dataset with undersampling of the FP samples. Each epoch had all TP samples and the same count of the FP samples.
I used batch sampling to provide a balanced batch of TP/FP samples - after each TP I added FP with the same species id.

In the loss function, I calculate loss only for those sigmoid outputs that present in train_tp.csv or train_fp.csv. So all noisy samples are ignored.
FP co-teaching training

I have tried to find a way how to use FOCI or SELFIE to work with noisy data, but all of them use historical predictions of each sample. With my random crop and frequency band mixup it's almost impossible. Even shift for 0.5-1 seconds can add a new species to the sample. So historical data will be incorrect.

I tried co-teaching training because it doesn't require historical data.
Paper: https://arxiv.org/pdf/1804.06872.pdf
Code sample: https://github.com/bhanML/Co-teaching

When I implemented co-teaching training I have only 5 days before the deadline.
The first experiments with co-teaching gave 0.830 LB for the TP and 0.880 LB for the FP. So it looked like a bad experiment.

I tried to improve loss function by adding high loss samples wit
h changed targets (by default co-teaching should ignore high los
s samples as wrong samples).

The final loss function consists of:

    50% lowest loss samples (all confident samples mandatory add
ed to this part of loss with scale factor 2)
    45% ignored losses
    5% highest loss samples with the changed target (1 for predi
ctions with sigmoid >= 0.5 and 0 for sigmoid < 0.5)
    The loss implementation is here: https://github.com/MPGek/mp
gek-rfcx-species-audio-detection-public/blob/main/model/forward_
passes_coteaching.py

When I invented this loss I had only 3 days before the deadline.
This loss has a good potential for future experiments. I trained
 only 2 experiments with this loss, both with the same hyperpara
meters. The first one had a bug so it produces extremely high me
trics and used wrong epochs to submission - however, even with t
he bug it produces a good LB score comparing to the original FP
training.
Folds and best epoch metric

To chose the best epoch in all training I used the BCE which cal
culated only on confident samples.
In some experiments I used 5 stratified KFold, in some, I used 7
 folds with stratified shuffle split with test size 0.3.
Ensembles

The TP training with EfficientNetB7 gave me only 0.912-0.915 on
the public LB.
The FP training with EfficientNetB2-B4 gave only 0.887-0.893 LB.
The ensemble of the TP and FP gave 0.929 LB.

The FP co-teaching training on simple EfficientNetB2 gave me 0.9
25 LB (a quite good improvement from the original FP with 0.893)

The final ensemble consists of all best experiments (0.941 publi
c LB and 0.944 private LB):

    TP EfficientNetB7 with 0.915
    FP EfficientNetB2-B4 with 0.893
    FP co-teaching EfficientNetB2 with 0.925

==> top23.txt <==
23rd Place Solution: Supervised Contrastive Learning Meets Domai
n Generalization (with TF code)
Posted in rfcx-species-audio-detection 7 days ago

24
Introduction

Thanks Kaggle for this exciting competition and our team (@dath
udeptrai @mcggood @akensert @theoviel @ratthachat) congratulate
 all winners -- we have learned a lot from this competition and
all winners¿ solutions !!

From the winner solutions, it turns out there are mainly 4 ways to break 0.95x

- Masked loss
- Post-processing
- Pseudo-labelling
- Extra labeling

We actually tried the first three but unfortunately could not really make them work effectively.
Here, as an alternative solution, we would love to share our own solution which is able to reach 0.943 Private.
Training pipeline (improve from 0.80x baseline to 0.92x)
Baseline

Our baseline models score slightly above 0.8. We adopt an audio tagging approach using a densenet121 backbone and the BCE loss.

Our training pipeline includes the following tricks :

- Class-balanced sampling, using 8 classes per batch. Batch sizes were usually 64, and 32 for bigger models.
- Cyclical learning rate with min_lr is 0.0001 and max_lr is 0.001, step_size is 3 epochs. We train models for 100 epochs with early stopping.
- LookAhead with Adam optimizer (sync_period is 10 and slow_step_size is 0.5)

Pretraining with Supervised contrastive Learning (SCL) [0.81x -> 0.85x]

Because of the small amount of data, models overfit quickly. To solve this problem,two options were using external data and cleverly pretraining our models. Unlike a lot of competitors, we focused on self-pretraining techniques : @dathudeptrai tried auto-encoders, GANs, SimCLR, Cola, and Supervised Contrastive Learning which ultimately was the only thing to work.
Non-overlap time Cutmix [0.85x -> 0.88x]

Our sampling strategy consists of randomly selecting a crop containing the label. Most of the time, crops are bigger than labels which introduces false positives. One idea to make full use of our windows was to adapt cutmix to concatenate samples such that labels are entirely kept (when possible).
Domain Generalization with MixStyle [0.88x -> 0.89x]

Domain shift always exists in deep learning, in both practice and kaggle challenges, especially for small data. Therefore, domain generalization techniques should help with robustness. We applied a simple yet effective technique called Mixstyle.
Multi Scale inference (MSI) [0.89x -> 0.91x]

Duration of species¿ call varies quite a lot. For example, for class 3 it is around 0.7 seconds while for class 23 is around 8 seconds. To use this prior information, we use multiple window sizes (instead of using a single one). For each class, we choose the one that yields the best CV. In case we have multiple window

sizes reaching the maximum, we take the largest window. Although our CV setup which consists of crops centered around the labels did not correlate really well with LB, the 2% CV improvement reflected on LB quite well.
Positive learning and Negative learning [0.91x -> 0.92x]

We used the following assumption to improve the training of our models :
For a given recording, if a species has a FP and no TP, then it is not in the call. Our BCE was then updated to make sure the model predicts 0 for such species.
Ensembling

Our best single model densenet121 scores around 0.92 public and 0.93 private. Averaging some models with different backbones, we were able to reach 0.937. We tried many different ensembling, scale fixing and post-processing ideas, and were able to improve our score a bit, but unfortunately we could not find the real magic.

In the end, we empirically analyzed the most uncertain class predictions from our best models, and averaged predictions with other (weaker) models. We relied on diversity to make our submission more robust. Our final ensemble scored public 0.942 and private 0.943.
Thanks for reading !

TensorFlow Code Here https://github.com/dathudeptrai/rfcx-kaggle

==> top25.txt <==
s3 class postprocessing (+0.008 on private LB)
Posted in rfcx-species-audio-detection 9 days ago

15

As it was mentioned by others, s3 was a very interesting class. Looking more into this, our team figured out that we should scale up predictions for s3 class. We checked a number of ways to do this, but eventually a simple scale of 1.5x worked best for us (our predictions were all positive numbers, that scaling probably wouldn't work if your predictions are in logits).

This scaling lifted our final ensemble from 0.933 to 0.941 (+0.008) on private LB.

The intuition to convince ourselves that this was not a small public LB fluke (because it similarly helped on public LB) was as follows:

Each class roughly has similar number of TP labels
But you can see on OOF train predictions and on test prediction that classes are very imbalanced (let's say looking at frequencies of different classes being top1 or top3 predictions). And s3 class is actually clearly the most ubiquitous.
Now because of specific labeling of this competition, we are in a situation where s3 is very common class, and hence very often present next to other labels, and model gets 0 target for s3 class (if you did not do any corrections, like loss masking)

So model learns to be extra cautious at predicting s3, when in fact it should be quite aggressive in predicting it

Hence we applied post-processing here.
We could not leverage this on other classes, but s3 really stood out as a clear outlier for us.
=== q ===
I did the same thing.
In my model, just scaling up the prediction for s3 improved it by 0.019 (0.898 -> 0.917) at private LB.
And scaling down s21, which was the next least accurate, improved the score by 0.001 (0.917 -> 0.918).
I did not explore further because I thought it was not essential.
I was guessing that only s3 had more training label errors (not confirmed), but reading your discussion helped me understand better. Thank you for sharing.

==> top27.txt <==
27th place simple solution (0.932 public, 0.940 private) - dl_pipeline
Posted in rfcx-species-audio-detection 9 days ago

23

In summary:

Best single model (0.925 public lb): densenet121 features + fastai head
Loss function: cross entropy
Sampling: 128x1024 crops around true positive samples
Spectrogram parameters: n mels 128, hop length 640, sample rate 32000
Augmentations: clipping distortion, pitch shift and mixup
Inference: Predict on crops with a small width (e.g. 128x128 instead of 128x1024 used for training) and calculate the max probability for each of the 24 classes.

Introductory monologue
First and foremost, this was an interesting competition and a good learning opportunity as it is often the case in Kaggle! One ¿problem¿ of this competition is that the test data was labeled with a different method and no samples of labeled test data were provided. This makes it difficult to get a sense of validation score and increases the danger of overfiting the public test results. In fact, I almost gave up on this competition when I realized this was the case. But eventually I decided to get back to it and work on a simple solution and on a python library ¿ dl_pipeline (https://github.com/mnpinto/dl_pipeline) ¿ that I will use as a general framework for future kaggle competitions in general. Initially, the idea for dl_pipeline was just to keep my code more organized and more reusable but I figured that maybe there¿s also some value in sharing it.

Data preprocessing
Save all wave files in npy files with sample rate of 32000 Hz to save time.

```
def audio2npy(file, path_save:Path, sample_rate=32_000):
    path_save.mkdir(exist_ok=True, parents=True)
    wave, _ = librosa.load(file, sr=sample_rate)
    np.save(path_save/f'{file.stem}.npy', wave)
```

I didn't convert the audio to spectrograms right away since I st
ill want the ability to use audio augmentation on the waveforms.

Augmentations and Spectrograms

    First I create crops on the waveform including the true posi
tive labels with a number of samples calculated so that the spec
trogram will have a width of 1024.

Note: Cropping before applying the augmentations is much faster
than the other way around.

    Then for the waveform augmentations I used the audiomentatio
ns library (https://github.com/iver56/audiomentations). I ended
up using just the following augmentations as based on public lb
I didn't find that others were helping, although this would requ
ire a proper validation to take any conclusions.

```
def audio_augment(sample_rate, p=0.25):
    return Pipeline([
        ClippingDistortion(sample_rate, max_percentile_threshold
=10, p=p),
        PitchShift(sample_rate, min_semitones=-8, max_semitones=
8, p=p),
    ])
```

Note: Some augmentations are much slower, for example, pitch shi
ft and time stretch. When using those augmentations the probabil
ity of use makes a big difference in how long the training takes
.

    Then I searched the fastest way to convert the audio to spec
trograms in the GPU and I ended up using nnAudio (https://github
.com/KinWaiCheuk/nnAudio). Again, converting to spectrogram afte
r the waveform is cropped is a nice gain in processing time.

Model
I tried several models but the one that got me a better result i
n the public leaderboard was densenet121 and the second-best Res
NeSt50. One particularity is that I use for all the models the f
astai head with strong dropout.

The fastai head (using create_head(num_features*2, num_classes,
ps=0.8)).

```
(1): Sequential(
      (0): AdaptiveConcatPool2d(
        (ap): AdaptiveAvgPool2d(output_size=1)
        (mp): AdaptiveMaxPool2d(output_size=1)
      )
      (1): Flatten(full=False)
      (2): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=Tru
```

```
e, track_running_stats=True)
      (3): Dropout(p=0.4, inplace=False)
      (4): Linear(in_features=2048, out_features=512, bias=False
)
      (5): ReLU(inplace=True)
      (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      (7): Dropout(p=0.8, inplace=False)
      (8): Linear(in_features=512, out_features=24, bias=False)
    )
```

Training
I guess code speaks more than words, particularly for those fami liar with fastai:

```
bs = 32
learn = Learner(dls, model, loss_func=cross_entropy, metrics=[ac
curacy, lrap], cbs=cbs)
learn.to_fp16(clip=0.5);
learn.fit_one_cycle(30, 1e-3, wd=3e-2, div_final=10, div=10)
```

So in English, this is a one cycle learning rate schedule with 3 0 epochs starting with lr=1e-4, increasing to 1e-3 and then decr easing back to 1e-4 following cosine anealing schedule. The loss function is the good old cross-entropy. Also, a weight decay of 3e-2 was used, a gradient clip of 0.5 and the train was done wi th mixed-precision so that my GTX 1080 can handle a batch size o f 32 with 128x1024 image size.

One training epoch takes about 1 minute on my GTX 1080, I guess it's not bad considering that I'm doing waveform augmentations o n CPU that even with p=0.25 take some time.

Inference
This is the fun part because it was almost by mistake that I rea lised that making inference with smaller tiles is way better. I presume that this is the case because I'm training with cross-en tropy for a single label problem but the test data is labelled w ith multiple labels. By using smaller crops the predictions are more multilabel friendly. The reason I've been using cross-entro py instead of binary cross-entropy and sigmoid for the typical m ultilabel problem is that for me the convergence was much faster using the cross-entropy approach and with better results. Maybe I made a mistake somewhere I don't know, I didn't investigate i t in much detail.

Run predictions on crops of the spectrogram with a width of 64, 128 and 256 (remember training was done with 1024), calculat e the max probability for each class for each case (64, 128, 256) and the average of the 3 cases. The average of the 3 gave me p ublic lb 0.928 on my best single model that I describe above, co mpared to 0.925 for just the 128 width inference.

The final solution with public lb 0.932 and private lb 0.940 is an ensemble of a few training interations with some modifica tions. (I will update this tomorrow with more information).

dl_pipeline

And again the code for this solution is now public on this repo:
 https://github.com/mnpinto/dl_pipeline

The following code should correspond to the best single model so
lution but I need to check if I didn't mess up anything when cle
aning the code:

```
#!/bin/bash
arch='densenet121'
model_name='model_0'
sample_rate=32000
n_mels=128
hop_length=640

for fold in 0 1 2 3 4
do
    echo "Training $model for fold $fold"
    kaggle_rainforest2021 --fold $fold --model_name $model_name
\
        --model $arch --sample_rate $sample_rate --n_mels $n_mel
s \
        --hop_length $hop_length --bs 32 --head_ps 0.8 \
        --tile_width 1024 --mixup true >> log.train
done

for tw in 64 128 256
do
    echo "Generate predictions for $model with tile_width of $tw
"
    kaggle_rainforest2021 --run_test true --model_name $model_na
me \
        --model $arch --sample_rate $sample_rate --n_mels $n_mel
s \
        --hop_length $hop_length --tile_width $tw \
        --save_preds true >> log.predict
done
```

== q ===
Thanks for sharing. I'll ask you this question but it is valid f
or all those who did not use the FP data. TP only gives you grou
nd truth of 1. What ground truth 0 did you all use?

I am quite puzzled to be honest.

Did you assume that the ground truth is zero for all other speci
es in each TP crop? If so then it is actually wrong, Some crops
have more than one species.

Anyway, I'd be happy to hear what you did for ground truth. And
congrats on the good result.
== a ==
You are correct @cpmpml, I considered 0 to all other species. I
worked on this as a single class classification problem using cr
oss-entropy loss for training. Initially I was using smaller aud
io crops (128x128 or 128x256) and the rationale was that probabl
y there's not much overlap of classes in the small crops around
the TPs (i.e. if I know class "A" is observed in that small crop
, it's quite likely that most, if any, of the other 23 won't be

in the same crop). And indeed with cross-entropy loss the model converges quite well. I now see that the idea of using BCE with TP as 1 and FP as 0 and masking all other values is what I was m issing and a great way to incorporate the FP. Nevertheless, with Chris Deotte post-processing my best single model gets 0.950 on private LB (resnest50 and resnest101), generating the predictio ns over crops of 128 (steps 64) and 256 (steps 128).
github: for top27 https://github.com/mnpinto/dl_pipeline

==> top32.txt <==
32nd place solution - what worked for me
Posted in rfcx-species-audio-detection 10 days ago

20

Congratulations to the top finishers!

This was my first encounter to audio competition, so I tried a l ot of maybe implausible ideas and learned a lot. Especially, sol utions from Cornell Birdcall Identification and Freesound Audio Tagging 2019 were helpful.

My finish is not strong, but I wanted to share some of the thing s that I believe(not sure since my score is not sufficiently hig h) worked for me (increased cv or public score), and hear what o ther kagglers experienced.

Frequency Crop

For one audio clip, crop 24 different crops according to fmin&fmax of each species.

I believe it is similar to what @cpmpml did and @hengck2 3 did(without repeating convolution computations)

LSoft

I used only TP and FP crops as labels. For example, for each row in TP or FP, only 1 out of 24 labels is present.

Based on BCELoss, I used 'lsoft' for unknown labels. LSo ft is introduced kindly by @romul0212 at https://github.com/lRom ul/argus-freesound/blob/master/src/losses.py

This is my loss computation with LSoft. mask indicates w here the label is known. true for unknown labels is initialized with 0.

```
tmp_true = (1 - lsoft) * true + lsoft * torch.sigmoid(pred)
true = torch.where(masks == 0, tmp_true, true)
loss = nn.BCEWithLogitsLoss()(pred, true)
```

Iterative Pseudo Training

Since train set is only very sparsely annotated, I thoug ht re-labeling with the model then re-training will help, and it indeed helped. I pseudo trained for 3 stages.

When pseudo training, I didn't use LSoft and used vanill a BCE.

LSEPLoss

Our metric is LWLRAP, so it is important to focus on rank between labels for each row. I used LSepLoss which fits this purpose, which is introduced kindly by @ddanevskyi at https://www.kaggle.com/c/freesound-audio-tagging-2019/discussion/97926

After stage3 of BCE pseudo training, I pseudo trained extra 2 stages with LSEPLoss

I fixed original code a bit to allow soft labels.

```
def lsep_loss(input, target):
    input_differences = input.unsqueeze(1) - input.unsqueeze(2)
    target_differences = target.unsqueeze(2) - target.unsqueeze(1)
    target_differences = torch.maximum(torch.tensor(0).to(input.device), target_differences)
    exps = input_differences.exp() * target_differences
    lsep = torch.log(1 + exps.sum(2).sum(1))
    return lsep.mean()
```

Global Average Pooling on only positive values

We need to know if the species is present or not. We don't care if it appears frequently or not. I thought doing global average pooling on whole last feature map of CNN will yield high probabilities for frequent occurrences of birdcall in one clip and low probabilities for infrequent occurrences, which doesn't match our goal. So I took mean of only positive values from the last feature map of CNN.

Following code is attached at the end of CNN's extracted feature map

```
mask = (x > 0).float()
features = (x*mask).sum(dim=(2, 3))/(torch.maximum(mask.sum(dim=(2, 3)), torch.tensor(1e-8).to(mask.device)))
```

Augmentations

Gaussian/Pink NoiseSNR, PitchShift, TimeStretch, TimeShift, VolumeControl, Mixup(take union of the labels), SpecAugment

Thanks to @hidehisaarai1213 for kindly sharing https://www.kaggle.com/hidehisaarai1213/rfcx-audio-data-augmentation-japanese-english

One inference on full clip

I didn't resize the spectrogram, so I was able to train on crops and infer on full image.

When we don't resize, due to the property of CNN, I believe doing sliding windows prediction on small crops is just an approximation for doing one inference on the full image.

Validation - only use known labels

I did validation clip-wise, only on TP and FP labels. From the prediction, I removed all values corresponding to unknown labels, flattened, then calculated LWLRAP. It correlated with LB quite well on my fold0

My baseline was not so strong(~0.8), so I might had fundamental mistakes in my baseline.
I achieved 0.927 public with efficientnet-b0 fold0 3seed average, but my score worsened when doing 5fold ensembling. I tried average, rank mean, scaling on axis1 then taking mean, calculating mean of pairwise differences taking average, but it didn't help.
I'm planning to study top solutions to find out what I missed

I'd really appreciate it if you share some opinions with my approaches and things that I missed.
=== q ===

you are given a partial label, i.e. you are told if a class is present in the tp. but you are not told if other classes are present. The opposite is for fp annotation, where we are told if a class is absent.

but we can create very confident negative samples. you can use external data or mix negative samples, etc.
the trick is to create a very large pool of negative samples.

because it is a ranking metric, we can score very well, if there is no negative sample in the top-3 or top-5.
The score of the positive samples can be low, but it should not be lower than the positive samples.

i think if you create many more negative samples, your score should improve (also for ensemble)

==> top38.txt <==
Hi all.
First, thanks for a great competition. Thats my favorite type of competitions where there are incomplete or noisy labels and you have to think how to deal with them.
Second, thanks to those who shared code. In particular to the authors of the following kernels:

https://www.kaggle.com/ashusma/training-rfcx-tensorflow-tpu-effnet-b2 - that was a great starter and I was just doing edits of that kernel to move on
https://www.kaggle.com/aikhmelnytskyy/resnet-tpu-on-colab-and-kaggle - that showed how you can train on colab as well

My code is in the following notebook - https://www.kaggle.com/vzaguskin/training-rfcx-tensorflow-tpu-effnet-b2-with-fp. The final submission is a merge of several versions of submissions from that code(and similar code on colab) plus s3 trick.

Now, how I got from initial 80+ in the starter notebook to 90+ and silver zone.

5-second cut worked better then initial 10 second
Added FP data with masked BCE/BCEFocal loss. I simply calcul

ate BCE loss only on the label I know is missing(and just usual BCE with label smoothing 0.2/Usual BCEFocal on TP data)
    Use heavier model(B4)
    Added mixup/cutmix

The best version of that code got .89+ on private LB.
Than ensembling goes - I just collect all the well scoring submissions(.87+ public) and average them. Ranking average seem to work slightly better than simple average(by 0.001 approximately).
The best private score I could get with that approach is .912

My version of s3 trick is that I multiply s3 by 2.5 and s7 by 2. That gave me .93 on private LB(.918 public). The version I selected for final had same .918 public and .929 private which is pretty much the same.

Again, thanks a lot for the competition. Learned many things and had a lot of fun.

Upd: I've added postprocessing from Chris and now this kernel scores 0.95467 private (gold zone) - which means complete training and inference on Kaggle TPU only within less than 3 hours and gold level score.

==> top43.txt <==
First of all, many thanks to Kaggler.
I got a lot of ideas from Kaggler in the discussion. And it was fun.

My solution summary is below.

    The high resolution of spectrogram
    Post-processing with moving average
    Teacher-student model for missing labels

1st stage: SED

I started from basic experiment with SED. Why SED? Because I think SED is strong in the multi label task.

I used log mel-spectrogram as the input of SED. Basic experiment involves data augmentation (Gaussian noise, SpecAugment and MixUP), backbone model choice and adjusting resolution of log mel-spectrogram. As a result, below condition was the best for me.

    No noise injection
    MixUp
    The best model architecture is EfficientNet
    The higher the resolution of log mel-spectrogram, the better the result.

The resolution

The most important one is the resolution. Recently, in Kaggle computer vision solution, the higher the resolution of the image, the better the result. In spectrogram, the same phenomenon may happen. In mel-spectrogram, the resolution can be changed by adjusting "hop_size" and "mel_bins". Following result is changing th

e resolution with ResNest50(single model).

Resolution(Width-Height)	public LB
1501-64(PANNs default)	0.692
3001-128	0.805
6001-64	0.725
1501-256	0.761
751-512	0.823
1501-512	0.821

The resolution was critical! According to experimental result, good resolution was "high" and similar to the square. 751-512 looks good. As a result, I chose 858-850. This configuration is as follows.

```
model_config = {
    "sample_rate": 48000,
    "window_size": 1024,
    "hop_size": 560,
    "mel_bins": 850,
    "fmin": 50,
    "fmax": 14000,
    "classes_num": 24
}
```

Post-processing

I used Framewise output for submission. It contains time and classes information. But there is a lot of false positive information in framewise output. Because they are not processing by a long time information. Therefore a short event of framewise output should be deleted. I prepared post-processing for framewise output. It is a moving average.

image.png

By taking a moving average in the time direction for each class, we can delete short events. This idea is based on the paper[1]. The sample code is as follows.

```
def post_processing(data): # data.shape = (24, 600) # (classes, time)
    result = []
    for i in range(len(data)):
        result.append(cv2.blur(data[i],(1,31)))
    return result
```

I improved LB by using moving average. The following result is comparing post-processing with EfficientNetB3(single model).

	public LB
w/o post-processing	0.785
w/ post-processing	0.840

Summary

- MixUp(alpha=0.1)
- Epoch 30
- Adam(lr=0.001) + CosineAnnealing(T=10)
- Batchsize 6
- Use only tp label

Get random tp clip 10 sec
The resolution of log mel-spectrogram: 858-850
Loss function: BCE
Weak label training
Post-processing: moving average

Then I got 0.916 public LB with EfficientNetB0(5-folds average ensemble).

2nd stage: missing labels and the ensemble

I reported discovering missing labels and re-train. And It didn't work. After that, I thought about missing labels again. My answer is that the model is not correct for discovering missing labels. There are a lot of missing labels around tp. Therefore the model is not correct.

train.png

To solve this issue, I used teacher-student model.

geretation.png

1st generation is similar to 1st stage. I gradually increased model prediction ratio. By using teacher-student model, I could discover missing labels. Specially, in strong label training, teacher-student model was effective. Following result is teacher-student model score with EfficientNetB0.

image.png

"MixUp rate" is probabilistic MixUp. This method is based on the paper[2].

Finally, I made the ensemble of 1st stage model and 2nd stage model. Ensemble procedure is simple average. Then I got 0.924 public LB.

References

[1] Teck Kai Chan, Cheng Siong Chin1 and Ye Li, "SEMI-SUPERVISED NMF-CNN FOR SOUND EVENT DETECTION".
[2] Yuan Gong, Yu-An Chung, and James Glass, "PSLA: Improving Audio Event Classification with
Pretraining, Sampling, Labeling, and Aggregation".

Appendix: the resolution and EfficientNet

Finally, I show interesting result. It is relationship between EfficientNet and the resolution. The following result is public LB(5-folds average ensemble).

Resolution(W-H)	751-512	751-751	858-850
EfficientNetB0	0.893	0.904	0.916
EfficientNetB3	0.913	0.912	0.900

In B0, the higher resolution, the better result. But B3 was vice versa. Usually, the larger EfficientNet, the better at high resolution it is. But the above is reverse. Why?

Maybe domain shift(train: noisy sound -> test: clean sound) is concerned. B3 has learned about train domain features(noisy sound

). On the other hand, B0 has less representational ability than B3. Therefore B0 has learned the common features of the train an d test domain with high resolution. Without domain shift, B3 wou ld have also shown good results with high resolution.

SEMI-SUPERVISED NMF-CNN FOR SOUND EVENT DETECTION

Technical Report

Teck Kai Chan,[1,2], Cheng Siong Chin[1]*

[1]Newcastle University Singapore
Faculty of Science, Agriculture, and Engineering
Singapore 599493
{t.k.chan2, cheng.chin}@newcastle.ac.uk

Ye Li[2]

[2]Xylem Water Solution Singapore Pte Ltd
3A International Business Park
Singapore 609935
ye.li@xyleminc.com

ABSTRACT

For the DCASE 2020 Challenge Task 4, this paper proposed a combinative approach using Nonnegative Matrix Factorization (NMF) and Convolutional Neural Network (CNN). The main idea begins with utilizing NMF to approximate strong labels for the weakly labeled data. Subsequently, based on the approximated strongly labeled data, two different CNNs are trained using a semi-supervised framework where one CNN is used for clip-level prediction and the other for frame-level prediction. Using this idea, the best model trained can achieve an event-based F1-score of 45.7% on the validation dataset. Using an ensemble of models, the event-based F1-score can be increased to 48.6%. By comparing with the baseline model, the proposed model outperforms the baseline model by a margin of over 8%.

Index Terms— Nonnegative matrix factorization, convolutional neural network, semi-supervised learning, DCASE 2020

1. INTRODUCTION

A Sound Event Detection (SED) system can be described as an intelligent system that is capable of not only detecting the types of sound events present in an audio recording but also returning the temporal location of the detected events. Such a system can be useful in several different domains and as compared to a visual detection system, it can be advantageous in several different aspects. Firstly, a SED system is not affected by the degree of illumination. Secondly, occluded objects do not affect detection accuracy. Thirdly, audio recording requires lesser computational resources as compared to an image or video. Finally, some events, such as a car horn, can only be detected by sound [1], [2].

However, for a SED system to achieve maximum performance, there may be a need for a large amount of strongly labeled data where the occurrence of each event with its onset and offset is known with certainty during the model development phase. This can be a limiting factor because such data is usually difficult and time-consuming to collect as it requires repeated listening and adjusting of label time boundaries on a visual interface [3].

As shown in our previous work [4], NMF can be used to approximate strong labels for the weakly labeled data. Thus, as a follow-up work, we proposed to label the weakly labeled data using NMF in a supervised manner. Using the approximated strongly labeled data, we then trained two different CNNs in a semi-supervised framework where one of the models will produce the clip level prediction. In contrast, the other model will produce a frame-level prediction.

Based on such an idea, our best model can achieve an event-based F1-score of 45.7% on the validation dataset. Using an ensemble of models, we can further increase the event-based F1-score to 48.6%. By comparing our models with the baseline model, our models outperformed the baseline model with a margin of over 8%.

The paper is organized as follow, Section 2 provides the information on the proposed methodology, section 3 provides the results and discussion, and finally, the paper ends with a conclusion.

2. PROPOSED METHODOLOGY

In this section, information on the proposed methodology will be provided in several different subsections.

2.1. Audio Preprocessing and Feature Extraction

As the first step of pre-processing, all audio recordings that were longer or shorter than 10s were first truncated or padded to have an equal length of 10s. The processed recordings were then resampled at 22,050 Hz, and spectrograms were tabulated for each recording using a Fast-Fourier Transform (FFT) window size of 2048 (92 ms)

* This work was supported by the Economic Development Board-Industrial Postgraduate Programme (EDB-IPP) of Singapore under Grant BH180750 with Xylem Water Solution Singapore Pte Ltd

with a hop length of 345 (15.6 ms). Mel-spectrograms were then tabulated using 64 mel filter banks. Based on such a setting, a tabulated mel spectrogram would have a size of 640 by 64, where 640 represents the number of frames, and 64 represents the number of mel bins. Finally, a logarithm operation was applied to obtain the log mel spectrogram, which will be used as input to the training model.

2.2. Approximating Strong Labels Using NMF

NMF is an effective multivariate data decomposition method popularized by Lee and Seung [5]. Given a nonnegative matrix V of size $m \times n$, the objective of NMF is to derive two nonnegative matrices, W of size $m \times r$ and H of size $r \times n$ such that V can be approximated by the linear combination of W and H. This can be formally be defined as

$$V \approx WH \quad (1)$$

Where W can be interpreted as the dictionary matrix and H can be interpreted as the activation matrix and r can be interpreted as the number of components.

As shown in our previous work [4], NMF can be used to approximate strong labels for the weakly labeled data. However, the methodology introduced in [4] can induce noise into the training data. With the availability of strongly labeled synthetic data, we proposed to approximate strong labels for weakly labeled data using a supervised approach before the calculation of log mel spectrogram.

The first step is to extract the event template from the mel spectrograms to form a dictionary for different event classes. Since the synthetic sound clip can contain multiple events, temporal masking was applied to the mel spectrogram using the given temporal annotations. Templates of each event class were then retrieved from the masked mel spectrogram using NMF by allowing r to be set as 1. For example, if synthetic clip A has Speech and Cat occurring at frame 1 to 100 and 100 to 110 respectively, all frames from 101 onwards were masked to extract the Speech template followed by masking all frames except frames 100 to 110 to extract the Cat template.

As weakly labeled data possessed the event tags, we then applied the corresponding dictionary on the sound clip to derive H. Frames that were activated (above a pre-defined threshold) were assumed to contain the event class. For example, if Clip B contains Speech and Dog, we first applied NMF to decompose Clip B using Speech dictionary and with r set as 1 to derive H. Frames that were over a threshold were then assumed to contain only Speech. A similar concept was applied to derive the temporal annotation for Dog by using the Dog dictionary instead of Speech dictionary.

2.3. Semi-Supervised Learning

As mentioned in [6], there can be a trade-off in SED performance due to the pooling operation. While the accuracy of clip level detection (also known as audio tagging) can be improved with higher temporal compression (pooling along the time axis), this can result in a degradation of accuracy in frame-level detection.

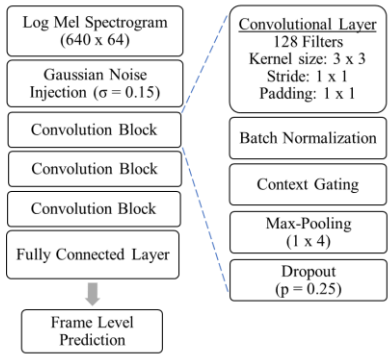


Fig. 1. Model for Frame Level Prediction

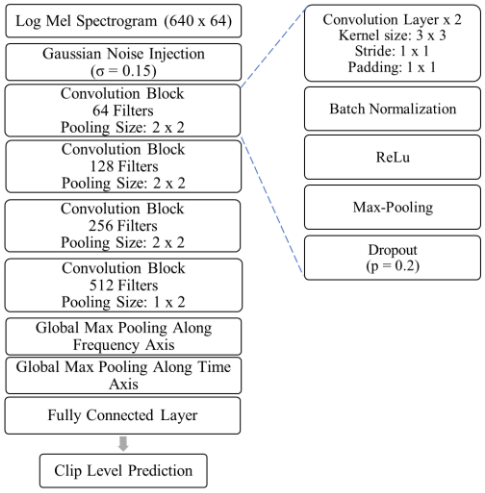


Fig. 2. Model for Clip Level Prediction

Therefore, we proposed a Shallow Model (SM) with no temporal compression for frame-level prediction and a Deep Model (DM) with temporal compression for clip level prediction. In addition to the difference in pooling size, SM has fewer convolutional layers, adopted context gating [7] as the activation function as opposed to ReLu and has a slightly higher dropout rate. The details of SM and DM can be found in Fig. 1 and Fig. 2 respectively.

Given that y_f and y_c are the frame level and clip level ground truth of an input respectively. The Binary Cross Entropy (BCE) loss between the frame-level prediction of SM, SM_f, and y_f can be given as

$$l_f = BCE(SM_f, y_f) \quad (2)$$

While the BCE loss between the clip level prediction of DM, DM_c, and y_c can be given as

$$l_c = BCE(DM_c, y_c) \quad (3)$$

As mentioned earlier, the accuracy of clip-level detection is better for models with higher temporal compression. We hypothesized that by enforcing the prediction of SM to be consistent with DM, it could produce a better frame-level prediction. As the prediction output of SM is in frame level, we applied a global max pooling on the time axis of SM_f to obtain the clip level prediction, SM_c. Instead of using BCE as the function, we proposed the use of Mean Square Error (MSE) as the consistency loss function. This was found to be a better consistency loss function as compared to using BCE [8].

$$l_{con} = MSE(SM_c, DM_c) \quad (4)$$

In addition, we also enforce the consistency of prediction on the unlabeled data. This is also known as semi-supervised learning, which was found to improve the performance and generalization of the model [9], [10]. Given that the frame level and clip level prediction from SM and DM on the unlabeled data are SM_{uf} and DM_{uc} respectively. The clip level prediction, SM_{uc}, from SM can be obtained by applying a global max pooling on SM_{uf}. Thus the consistency cost on the unlabeled data can be given as

$$l_{unlabel} = MSE(SM_{uc}, DM_{uc}) \quad (5)$$

However, if $l_{unlabel}$ was given too much weightage during the early stage of training, it may result in a degenerate solution where no meaningful classification of the data can be obtained [8]. Thus, a weighting parameter, w, is required to regularize the contribution of $l_{unlabel}$ throughout the entire training process. Following [8], w was proposed to ramp up from 0 along a Gaussian curve and can be defined as

$$w = \exp\left(-5\left(1-T\right)^2\right) \quad (6)$$

Where T is a positive value which represents the training progression. As an additional measure to prevent obtaining a degenerate solution, we proposed allowing $l_{unlabel}$ to be calculated if DM is confident with its prediction. Thus $l_{unlabel}$ is defined as

$$l_{unlabel} = \begin{cases} w \times MSE(SM_{uc}, DM_{uc}), \max(DM_{uc}) > \lambda \\ 0, otherwise \end{cases} \quad (7)$$

Where λ represent the level of confidence. Thus, the total combined loss, l_{total} can then be defined as

$$l_{total} = l_f + l_c + l_{con} + l_{unlabel} \quad (8)$$

Based on the calculated l_{total}, model parameters of both models will then be updated using Adam with its default parameters [11]. As it was found that the performance of deep NN may benefit from resetting the Learning Rate (LR) during the training process [12], we proposed to anneal the LR according to a cosine function and reset it to original LR after a certain number of epochs. The LR at each iteration is defined as [12]

$$LR_{curr} = LR_{\max} + \frac{1}{2}\left(LR_{\max} - LR_{\min}\right)\left(1+\cos\left(\frac{T_{curr}}{T_i}\pi\right)\right) \quad (9)$$

Where $LR_{\max}$ represents the maximum LR and was set as 0.0012. $LR_{\min}$ represents the minimum LR which was set as 1e-6. T_{curr} represents the current training iteration and T_i represent the maximum training iterations before a LR reset. If T_{curr} is equal to 0, the LR will be at its maximum value. If T_{curr} is equal to T_i, the LR rate will be at its minimum and at the next iteration, T_{curr} will then be reset to 0 while T_i is multiplied with an integer, T_{mult} which can delay the next restart if T_{mult} is larger than 1.

As T represents the training progression, which directly affects the calculation of $l_{unlabel}$. We proposed to define T as

$$T = \frac{T_{curr}}{T_i} \quad (10)$$

Thus the contribution w will be reset to 0 whenever the LR is reset.

Finally, we also adopted the concept of transfer learning in our system, where the models will first be trained using synthetic data for 5 epochs without the inclusion of $l_{unlabel}$. Only from the 6th epoch onwards, the parameters will be updated using real data and with the inclusion of $l_{unlabel}$.

2.4. Post Processing

A clip is considered to contain a specific event if the predicted probability from DM is larger than 0.5. Using the identified audio tag, temporal location can be found by locating the activated frames based on the predicted outputs from SM.

Before locating the activated frames, we smoothed the outputs from SM using iterative median filter [13] with an event-specific window size. Frames were then considered to be activated if they exceeded an event-specific frame threshold.

Following the implementation in [14], neighboring frames were also considered to be activated if they exceeded a lower bound threshold of 0.08. In addition, detected events with a duration of shorter than 0.1s were removed as they were considered as noise. Finally, we concatenated two similar events together if the difference between the first event offset and the second event onset is shorter than 0.2s.

	Event-Based F1-Score (%)	**PSDS F-Score (%)**	**PSDS**	**PSDS Cross Trigger**	**PSDS Macro**
PS 1 (T_i = 1 epochs, T_{mult} = 2, $LR_{\min}$ = 1e-6) *	45.2	63.6	0.630	0.548	0.408
PS 2 (T_i = 1 epochs, T_{mult} = 2, $LR_{\min}$ = 1e-6, trained with Synthetic Data) *	45.7	65.6	0.635	0.546	0.409
Ensemble System (PS 1 + PS 2) *	48.0	66.6	0.652	0.577	0.430
Ensemble System (PS 1 + PS 2) with Tuned MF Window *	48.6	66.5	0.649	0.573	0.425
Baseline without Source Separation	34.8	60.0	0.61	0.524	0.433
Baseline with Source Separation	35.6	60.5	0.626	0.546	0.449

Table 2. Results of Proposed Methodology Against Baseline Systems (PS refers to Proposed System and system with * were the submitted system to the DCASE Challenge Task 4)

3. RESULTS AND DISCUSSION

In our experiment, several factors can influence the training process and eventually affect the detection accuracy. Firstly, we found that if λ was set as a low value (i.e. 0.5), this will produce a higher $l_{unlabel}$, which resulted in a suboptimal solution. Thus, λ was set as 0.9 to ensure that $l_{unlabel}$ will only be calculated based on highly confident prediction.

Event	**Frame Threshold**	**First MF Window Size**	**Second MF Window Size**
Speech	0.3	7	15
Dog	0.3	7	15
Cat	0.3	3	6
Alarm/Bell Ringing	0.4	8	21
Dishes	0.2	3	5
Frying	0.6	24	48
Blender	0.6	3	6
Running Water	0.6	4	13
Vacuum Cleaner	0.4	24	48
Electric Shaver/ Toothbrush	0.4	48	96

Table. 1. Optimal Global Event Specific Threshold and MF Window Size (in Frames)

Secondly, by using an event-specific frame threshold, detection accuracy can be raised. However, optimal values differ across systems. Likewise, the median filter window is also dependent on the system trained. In our experiments, we found that using a smaller window in the first round of filtering and larger window size in the second round of filtering usually produces higher detection accuracy. Through multiple experiments, we derived the optimal global value for the event-specific frame threshold and filter window, and these are shown in Table 1.

In [14], the post-processing method was to join similar events together before the removal of noise. However, we realized that the accuracy could be higher if the noise were removed before the concatenation of similar events.

T_i is a hyperparameter that controls how fast LR will reduce from $LR_{\max}$ to $LR_{\min}$. In our experiment, if T_i was a small value (i.e. smaller than 5 epochs), T_{mult} must be at least 2 to prevent the large fluctuation of LR throughout the training process. Whereas if T_i was a large value (more than 5 epochs), T_{mult} can be set as 1 as the transition of LR from $LR_{\max}$ to $LR_{\min}$ can be considered slow and steady. Subsequently, we found that it is not a guarantee that a better solution can be found following a LR reset, and there is a possibility that a worse solution is found.

For transfer learning, we tested a different number of epochs for the transition of training data from synthetic data to real data. Based on our results, changing the data type after 5 epochs appeared to yield the best results.

Based on our methodology described in Section 2 and the aforementioned findings, our system can achieved an event-based F1-score of 45.2% by allowing T_i to be set as 1 epoch and T_{mult} as 2.

As mentioned earlier, models were trained using only synthetic data for the first 5 epochs and only from the 6th epoch onwards, model were trained using real data. In our experiment, we also tested a different form of transition where model were trained using synthetic data for the first 5 epoch whereas from 6th epoch onwards, models were trained using both real and synthetic data. Based on such setting, our system can achieved a slightly higher event based F1-score of 45.7% by allowing T_i to be set as 1 epoch and T_{mult} as 2.

Using an ensemble of the two models, we can further increase the event-based F1-score to 48.0%. As mentioned earlier, each system can have a different optimal MF window, we then tuned the MF window of the ensemble model, and this could further increase the event-based F1-score to 48.6%. However, we suspect this can have an adverse effect on the system, which may overfit the system to the validation dataset.

Based on the results shown in Table 2, all of the models trained using our proposed methodology were able to win the baseline system by a margin of at least 8%. We also computed the Polyphonic Sound Detection Score

(PSDS) [16] as a secondary measure. While we have a higher event-based F1-score, our system has a lower PSDS Macro score. It could be due to the difficulty of detecting Dishes and had a much lower detection accuracy (<20% across the systems) as compared to the other event class.

4. CONCLUSION

In this paper, a combinative approach using NMF and CNN was proposed for DCASE Challenge 2020 task 4. The proposed system could achieve an event-based F1-score of 45.7%, and with the use of the ensemble method, event-based F1-score raised to 48.6%. Based on such results, our system could outperform the baseline by a margin of over 8%. For our future work, we will investigate the cause of low detection accuracy for Dishes and improve our system in this aspect.

5. ACKNOWLEDGMENT

We would like to thank Nicolas Turpault for his technical support and also Kong Qiuqiang and Lin Liwei for their prompt replies in regards to our questions on their system. Finally, we would also like to express our gratitude to Giacomo Ferroni for his explanation on the PSDS metric.

6. REFERENCES

[1] Q. Kong, Y. Xu, I. Sobieraj, W. Wang, and M. D. Plumbley, "Sound Event Detection and Time–Frequency Segmentation from Weakly Labelled Data," *IEEE/ACM Trans on Audio, Speech, and Language Process.*, vol. 27, no. 4, pp. 777-787, Apr. 2019.

[2] Q. Zhou, Z. Feng and E. Benetos "Adaptive Noise Reduction for Sound Event Detection Using Subband-Weighted NMF," *Sensors*, vol. 19, no. 3206, pp. 1-19, Jul. 2019.

[3] B. Kim and B. Pardo, "Sound Event Detection Using Point-Labeled Data," *2019 IEEE Workshop on Appl. of Signal Process. to Audio and Acoustics*, New Paltz, New York, USA, Oct., 2019, pp. 1-5.

[4] T. K. Chan, C. S. Chin and Y. Li, "Nonnegative Matrix Factorization-Convolutional Neural Network (NMF-CNN) For Sound Event Detection," in *Proc. Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019)*, New York, USA, Oct. 2019, pp. 40-44.

[5] D. D. Lee, and H. S. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788-791, Oct. 1999.

[6] L. Lin, X. Wang, H. Liu and Y. Qian, "Guided Learning Convolution System For DCASE 2019 Task 4," in *Proc. Detection and Classification of Acoustic Scenes and Events 2019 Workshop*, New York, USA, Oct. 2019, pp. 134-138.

[7] A. Miech, I. Laptev and J. Sivic, "Learnable pooling with Context Gating for video classification," *arXiv preprint arXiv: 1706.06905*, pp. 1-8, Mar. 2018.

[8] S. Laine and T. Aila, "Temporal Ensembling For Semi-Supervised Learning," in *Proc. 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 2017, pp. 1-13.

[9] A. Oliver, A. Odena, C. Raffel, E. D. Cubuk and I. J. Goodfellow, "Realistic Evaluation of Deep Semi-Supervised Learning Algorithms," in *Proc. 32nd Conf. on Neural Information Process*. Syst. (NeurIPS 2018), Montreal, Canada, Dec. 2018, pp. 1-12.

[10] D. H. Lee, "Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks," in *Int. Conf. Mach. Learning 2013 Workshop : Challenges in Representation Learning (WREPL)*, Georgia, USA, Jun. 2013.

[11] D. P. Kingma and J. L. Ba, "ADAM: A Method For Stochastic Optimization," in *Proc. 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, USA, May 2015, pp. 1-15.

[12] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent With Warm Restarts," in *Proc. 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 2017, pp. 1-16.

[13] E. A. Castro and D. L. Donoho, "Does Median Filtering Truly Preserve Edges Better Than Linear Filtering?," *The Annals of Statistics*, vol. 37, no. 3, pp. 1172–1206, Apr. 2009.

[14] Q. Kong, Y. Cao, T. Iqbal, Yong Xu, W. Wang, and M. D. Plumbley, "Cross-task learning for audio-tagging, sound event detection spatial localization: DCASE 2019 baseline systems," *arXiv preprint arXiv: 1904.03476*, pp. 1-5.

[15] A. Mesaros, T. Heittola and T. Virtanen, "Metrics for Polyphonic Sound Event Detection," *Applied Sciences*, vol. 6, no. 162, pp. 1-17, May 2016.

[16] C. Bilen, G. Ferroni, F. Tuveri, J. Azcarreta and S. Krstulovic, "A Framework For The Robust Evaluation of Sound Event Detection," in *2020 IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, Barcelona, Spain, May 2020, pp. 61-65.

Ecological Informatics 59 (2020) 101113

Contents lists available at ScienceDirect

Ecological Informatics

journal homepage: www.elsevier.com/locate/ecolinf

A pipeline for identification of bird and frog species in tropical soundscape recordings using a convolutional neural network

Jack LeBien[a,*], Ming Zhong[b], Marconi Campos-Cerqueira[a], Julian P. Velev[c], Rahul Dodhia[b], Juan Lavista Ferres[b], T. Mitchell Aide[a,d]

[a] Sieve Analytics, San Juan, PR 00911, USA
[b] AI for Good Research Lab, Microsoft, USA
[c] Department of Physics, University of Puerto Rico, San Juan, PR 00931, USA
[d] Department of Biology, University of Puerto Rico, San Juan, PR 00931, USA

ARTICLE INFO

Keywords:
Acoustic monitoring
Bioacoustics
Sound classification
Convolutional neural network
Deep learning

ABSTRACT

Automated acoustic recorders can collect long-term soundscape data containing species-specific signals in remote environments. Ecologists have increasingly used them for studying diverse fauna around the globe. Deep learning methods have gained recent attention for automating the process of species identification in soundscape recordings. We present an end-to-end pipeline for training a convolutional neural network (CNN) for multi-species multi-label classification of soundscape recordings, starting from raw, unlabeled audio. Training data for species-specific signals are collected using a semi-automated procedure consisting of an efficient template-based signal detection algorithm and a graphical user interface for rapid detection validation. A CNN is then trained based on mel-spectrograms of sound to predict the set of species present in a recording. Transfer learning of a pre-trained model is employed to reduce the necessary training data and time. Furthermore, we define a loss function that allows for using true and false template-based detections to train a multi-class multi-label audio classifier. This approach leverages relevant absence (negative) information in training, and reduces the effort in creating multi-label training data by allowing weak labels. We evaluated the pipeline using a set of soundscape recordings collected across 749 sites in Puerto Rico. A CNN model was trained to identify 24 regional species of birds and frogs. The semi-automated training data collection process greatly reduced the manual effort required for training. The model was evaluated on an excluded set of 1000 randomly sampled 1-min soundscapes from 17 sites in the El Yunque National Forest. The test recordings contained an average of ~3 present target species per recording, and a maximum of 8. The test set also showed a large class imbalance with most species being present in less than 5% of recordings, and others present in > 25%. The model achieved a mean-average-precision of 0.893 across the 24 species. Across all predictions, the total average-precision was 0.975.

1. Introduction

Acoustic monitoring has gained widespread interest as an ecological tool for wildlife population assessment, conservation, and biodiversity research. Many species emit regular vocalizations or other acoustic signals that are species-specific, which enables monitoring via sound recognition. Advances in automated acoustic recorders have reduced prices and enabled data collection for months at a time. This has resulted in enormous data sets; however, like the evolution of many data-driven approaches, including camera traps and eDNA, methods for data collection are progressing faster than those for effective analysis and interpretation. In many cases, acoustic analyses are done manually, and this often limits the analyses to a subset of the complete datasets (Potamitis et al., 2014; Priyadarshani et al., 2018; Swiston and Mennill, 2009). To enable analysis of entire datasets, accurate, automated sound recognition methods are paramount.

Efforts to automate species identification in audio recordings have spanned diverse fauna such as birds, amphibians, bats, insects, fish and marine mammals (e.g. Ganchev, 2017). Recently, machine learning approaches have been successfully applied to acoustic data for identifying multiple species (Priyadarshani et al., 2018). Deep learning models such as convolutional neural networks (CNNs) have achieved remarkable performance (Florentin et al., 2020; Kahl et al., 2019; Ruff et al., 2019). Deep learning models have the advantage of incorporating

* Corresponding author.
E-mail address: jlebien@uno.edu (J. LeBien).

https://doi.org/10.1016/j.ecoinf.2020.101113
Received 29 January 2020; Received in revised form 14 April 2020
Available online 08 June 2020

feature learning in the training process, which eliminates or reduces the manual feature selection required. They are also often capable of scaling to a high number of classes. For example, in the 2016 BirdCLEF challenge, models were trained to recognize 999 bird species and evaluated on both omnidirectional soundscapes (i.e. ambient field recordings) containing multiple species and monodirectional recordings targeting single species (Goëau et al., 2016). CNNs achieved a significant performance increase in the challenge, compared to other methods that were mostly based on nearest neighbors or decision trees (Goëau et al., 2016; Sprengel et al., 2016). CNNs are a type of deep neural network that have achieved state-of-the-art performance on many image-recognition tasks (Aloysius and Geetha, 2017). Their network structure is characterized by layers designed for spatially-invariant image feature extraction. CNNs have also been successfully applied to many other sound recognition tasks, including bioacoustic recognition for taxonomic groups other than birds, and human voice recognition (Colonna et al., 2016; Kao et al., 2018). For sound recognition, CNNs typically use audio spectrograms as input, whereas mel-frequency cepstral coefficients (MFCCs) and other spectral statistics are common for other sound recognition models (Priyadarshani et al., 2018). Single CNNs can model many classes, as demonstrated in BirdCLEF and benchmark image recognition tasks (e.g. ImageNet, Deng et al., 2009) (Goëau et al., 2016; He et al., 2016). This is of particular interest for biodiversity monitoring efforts with many target species. These points highlight the potential of CNNs as an important tool for the ecological community.

Sprengel et al. (2016) used a six-layer single-label CNN trained on audio spectrograms to classify audio recordings targeting foreground bird species. They used data augmentation techniques such as combining same-class audio samples and samples of noise. Ruff et al. (2019) collected training samples directly from soundscape recordings and implemented a six-layer convolutional neural network for detection and classification of six owl species, incorporating a noise class to account for audio containing no target species. Florentin et al (2019) repurposed several popular pre-trained CNN image classifiers for the detection and classification of birds in soundscape recordings. They also used a noise class and, notably, found that including previously-identified false detections of target signals in the noise class improved performance. They also found that deeper network architectures generally performed the best on field (soundscape) data. Incze et al. (2018) repurposed a pre-trained MobileNet CNN (Howard et al., 2017) for single-species classification of bird audio recordings, and found that mapping grayscale input spectrograms to a color scheme improved performance (Incze et al., 2018). Other studies have found that combining various types of spectrograms into 3-channel images have improved performance for CNNs pre-trained for image classification (Sevilla et al., 2017; Xie et al., 2018; Xie et al., 2019). Sevilla et al. (2017) achieved relatively high-performance classification of single-species recordings and soundscapes by using an Inception-type CNN architecture (Szegedy et al., 2015) with time and time-frequency attention mechanisms. These studies have used single-label models for classification, and aggregated predictions over segments of soundscape recordings to make multi-label predictions. However, other studies have reported improved performance in soundscape classification with multi-label prediction models (Kahl et al., 2017; Zhang et al., 2016). As soundscape recordings can contain simultaneous occurrences of different target signals, multi-label prediction is a natural choice. However, the number of multi-label soundscape classification studies is limited due to the increased difficulty of acquiring multi-label training data. The best-performing team of the 2019 BirdCLEF challenge, which focused on soundscape recordings containing multiple species, used pre-trained ResNet (He et al., 2016) and Inception-type CNN models repurposed for classification of mel-scaled spectrograms of audio clips, and achieved the best performance with ensembled single-label and multi-label models (Kahl et al., 2019). Various data augmentation techniques were also found to significantly improve performance, such as time and frequency shifting and stretching of target signals, and adding Gaussian noise or noise from soundscapes (Koh et al., 2019; Lasseck, 2019). This competition's training data consisted primarily of recordings targeting single foreground species, but also a smaller validation set of annotated soundscapes like those in the test set. Results from this challenge showed a significant increase in performance for submissions that incorporated the validation soundscape data in training (Kahl et al., 2019). Specifically, Lasseck (2019) found that adding background noise from the validation soundscapes to training samples significantly improved performance on the test soundscapes. This indicates that acoustic monitoring systems can benefit significantly from location-specific training data.

Several challenges remain to be addressed for effective application of deep learning in acoustic monitoring. First, CNNs often require many training samples for each class. Large-scale species recognition efforts, such as for the BirdCLEF challenge, often use crowd-sourced public training data from various geographic sources (e.g. Xeno-Canto dataset, Vellinga, 2020;), and thus far these public datasets focus on birds and consist mainly of recordings targeting single foreground species. However, as mentioned above, geographic variation in soundscapes could require training data collected at a local or regional scale for high performance. This, however, would greatly increase the data labeling effort. Second, the non-directional nature of soundscape recordings demands accurate detection as well as classification for species recognition. Existing studies have demonstrated the difficulty of species recognition in soundscapes, compared to recordings that target a single foreground signal (Goëau et al., 2015; Goëau et al., 2016; Goëau et al., 2017; Goëau et al., 2018; Kahl et al., 2019). While many studies have evaluated models for classification of prior signal detections, methods that integrate both detection and classification of multiple species in noisy soundscapes are scarcer (Priyadarshani et al., 2018). Soundscape recordings often contain multiple species with calls overlapping in time and frequency, and a variety of environmental noises. Thus, models are challenged with achieving accurate multi-label recognition of simultaneous sounds, and a low false detection rate despite a variety of input noise. Third, although multi-label prediction is naturally desired for soundscapes containing different target signals that frequently overlap in time, collection of multi-label training data is typically more difficult than single-label data due to the demand of fully-labeling all target signals in each sample. Furthermore, class imbalance can be an issue for multi-label data because the number of positive instances of each class is more difficult to control than single-label data.

Considering these challenges, methods that mitigate region-specific training data collection effort and optimize model precision (minimizing false positives) in soundscape recordings are desired. To address this, we developed a pipeline for training CNNs for multi-species multi-label classification of soundscape recordings using single-label true and false-positive detections of each species. For the purpose of training data collection, we created an efficient implementation of a template-based sound detector and a graphical user interface for post-detection validation. A custom training loss was used to enable multi-label learning from single-label training data, and incorporate false-positive detections into training, which improved model precision in the presence of simultaneous and frequency-overlapping call types. This approach to model building: 1) mitigates the manual effort in collecting multi-label training data directly from soundscapes, 2) incorporates relevant absence information for each class by using false-positives, and 3) allows for controlling the number of positive and negative examples of each class in multi-label learning. In a related study (Zhong et al., 2020), we evaluated several CNN architectures and training methods using 2-s single-labeled audio clips for classification. In this paper, we apply the best-performing method to multi-label classification of soundscape recordings, and describe an end-to-end pipeline for model creation, starting from unlabeled soundscapes. The template-based detection and validation tools were used to generate the CNN training dataset in a semi-automated fashion, and greatly reduced the manual

effort required. ResNet50 transfer learning was applied to reduce the necessary training data and time. High precision and recall were achieved in predicting the presence of 24 tropical bird and frog species from 1-min soundscape recordings in the El Yunque National Forest. The presented pipeline is highly generic and expected to be applicable to many call types from diverse habitats.

2. Methods

2.1. Data acquisition

Soundscape recordings previously collected throughout Puerto Rico for other projects were used to create training and test datasets. AudioMoth recorders (Hill et al., 2018) were used to collect acoustic data. Recorders were placed on trees at the height of 1.5 m and programmed to record 1 min of audio every 10 min for a total of 144 recordings per day at a sampling rate of 48 kHz. A small portion (~10%) of recordings were sampled at 44.1 kHz. All recordings were stored in the ARBIMON web-based platform (Aide et al., 2013). In total, 97,900 1-min soundscape recordings were weakly or fully annotated with species-specific time-frequency bounding boxes as detailed in the next section. These recordings came from 749 sites across the island, however, 49% of recordings with annotations came from 152 sites throughout the El Yunque National Forest (Fig. 1). Recordings from other sites and older years were searched to increase the training sample size for certain species. 78% of recordings were collected in 2019, 11% in 2018, and 11% collected between 2015 and 2018. Most recordings were collected in the months March and April, a period of high acoustic activity.

One thousand recordings collected in April 2019 were randomly selected from 17 sites in the El Yunque National Forest, and used as a test set. The 17 sites were selected to cover species-rich habitats, and various elevations and monitoring transects throughout the forest.

2.2. Training data collection

Twenty-four species of bird and amphibian were chosen to be included in the CNN sound recognition model (Table 1). The species in this study included species of "Great Conservation Need" according to the State Wildlife Action Plan (Puerto Rico State Wildlife Action Plan, 2015) and regionally common species (e.g., *Eleutherodactylus coqui, Margarops fuscatus, Turdus plumbeus, Patagioenas squamosa*). For each species, examples of the one or two most common call types were collected for the purpose of CNN training (Fig. 2). To collect CNN training data for each species, we applied a template-based signal

Table 1
Numbers of true and false-positive template-based detections used for model training.

Species	Abbreviation	Taxon	True positives (*tp*)	False positives (*fp*)
Eleutherodactylus unicolor	ELUN	Frog	18,810	2209
Eleutherodactylus brittoni	ELBR	Frog	9369	6003
Eleutherodactylus wightmanae	ELWI	Frog	7690	10,222
Eleutherodactylus coqui	ELCO	Frog	6007	1966
Eleutherodactylus hedricki	ELHE	Frog	4687	10,318
Eleutherodactylus gryllus	ELGR	Frog	3682	8718
Eleutherodactylus richmondi	ELRI	Frog	3327	10,087
Eleutherodactylus portoricensis	ELPO	Frog	2550	4087
Eleutherodactylus locustus	ELLO	Frog	2492	2523
Eleutherodactylus antillensis	ELAN	Frog	1513	8685
Leptodactylus albilabris	LEAL	Frog	812	6505
Vireo altiloquus	VIAL	Bird	7745	9942
Loxigilla portoricensis	LOPO	Bird	3391	15,804
Patagioenas squamosa	PASQ	Bird	2162	2276
Spindalis portoricensis	SPPO	Bird	2024	10,080
Nesospingus speculiferus	NEES	Bird	1771	9226
Megascops nudipes	MENU	Bird	1634	3100
Margarops fuscatus	MAFU	Bird	1576	8054
Setophaga angelae	SEAN	Bird	1261	11,099
Turdus plumbeus	TUPL	Bird	976	7466
Melanerpes portoricensis	MEPO	Bird	932	26,290
Todus mexicanus	TOME	Bird	906	6143
Coereba flaveola	COFL	Bird	859	1748
Coccyzus vieilloti	COVI	Bird	476	6357

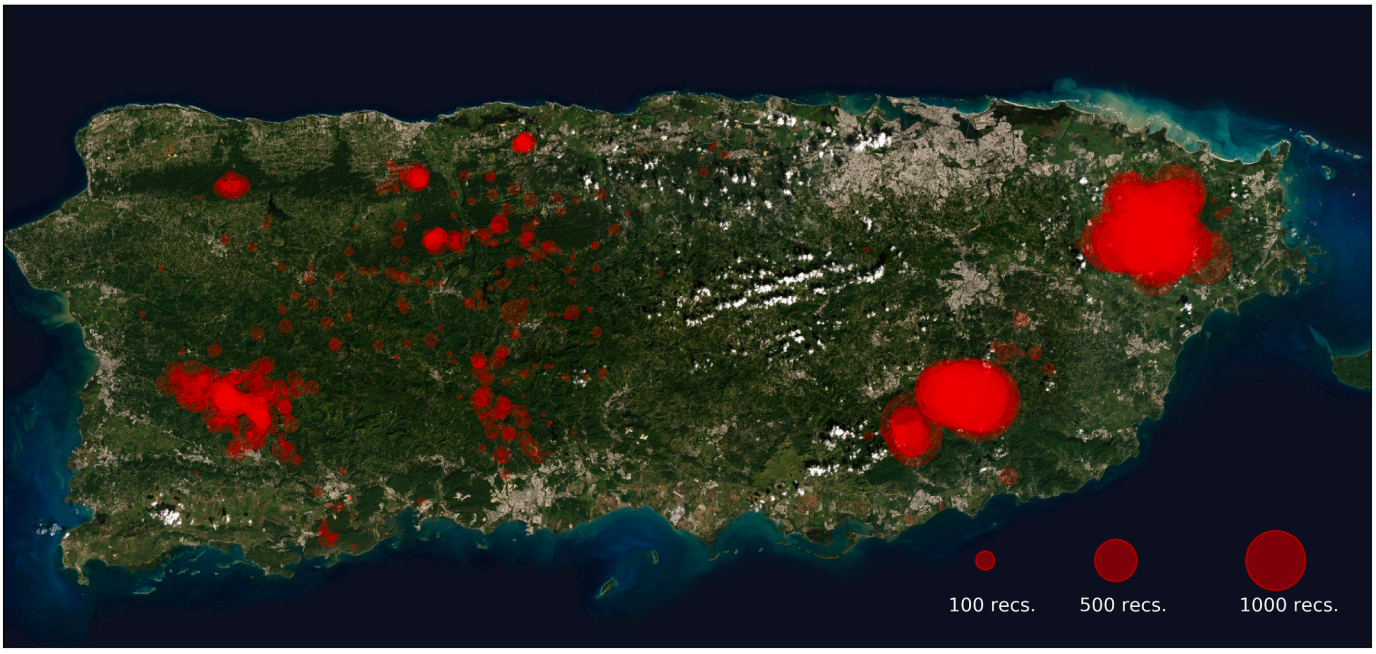


Fig. 1. Spatial distribution of recordings used for training and testing. Circles represent recorder sites with diameter indicating the number of recordings used. Most recordings came from El Yunque National Forest (large cluster in upper right) and the test recordings were sampled from this area.

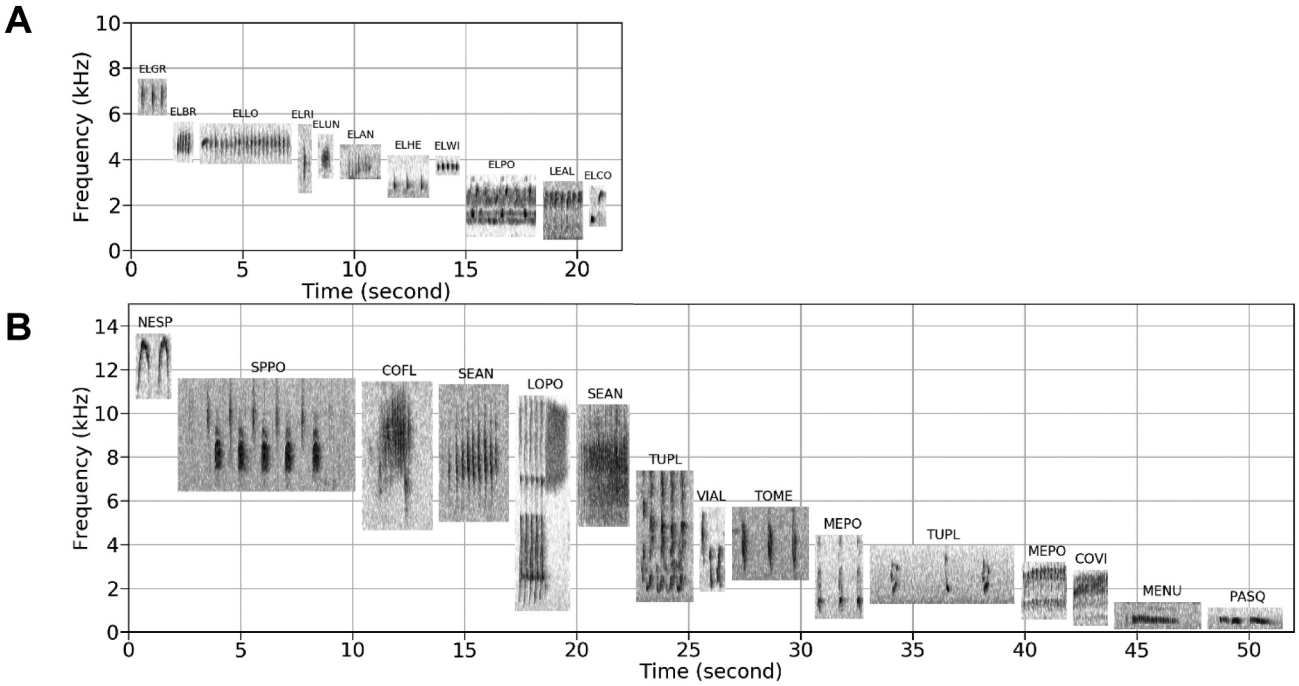


Fig. 2. Templates for each target species of frog (A) and bird (B). Species abbreviations are the same as in Table 1. The spectrograms show the pattern of acoustic energy in time and frequency of each call type, with darker values indicating higher amplitude. The template position with respect to the vertical axis indicates the frequency range of the call. The horizontal extent of the templates represents their time duration. The axes of the two plots are shown in proportion.

detection algorithm. This process consisted of three primary steps:

2.2.1. Template creation

Based on prior knowledge of the target species' sound repertoires, one or two of the most common call types were chosen as template signals for each species. The ARBIMON Visualizer feature was used to search audio spectrograms for a high signal-to-noise-ratio example of each call type. Time-frequency bounding boxes were drawn around the identified template calls and labeled by species and call-type. This metadata was stored in the platform for later analyses.

2.2.2. Template match detection

Using the ARBIMON Pattern Matching feature, developed as part of this study, each template was used to search through a playlist of recordings for calls that matched the template. This procedure detects time-localized signals with a correlation equal to or greater than a threshold assigned by the user. The correlation is computed in the spectrogram (time-frequency) domain.

The function *spectrogram* from the Python package *SciPy* is used for spectrogram generation (Oliphant, 2007). Spectrograms are computed from Hann-windowed 512-sample (~10 ms) segments of audio data, with 50% overlap between segments, and 512 Fast Fourier Transform (FFT) coefficients per segment. Time-frequency bin amplitudes are log-scaled. Note that these spectrogram parameters used for template match detection are different from those used to create the CNN inputs, which are described below.

The stored time-frequency coordinates of the template are used to extract it from its source recording. If necessary, playlist recordings are re-sampled to the sampling rate of the template's source recording. For each spectrogram in the playlist, frequency bins outside the template's frequency range are discarded, resulting in an image height equal to that of the template. For each remaining frequency bin, the median intensity over time is subtracted. This “flattening” step reduces the influence of acoustic processes that are approximately stationary throughout the recording (e.g. some insects, rain) in the search for time-localized signals (e.g. vocalizations).

The normalized cross-correlation (NCC) is then computed between the template T and the cropped, flattened spectrogram S, defined as follows:

$$\gamma(u) = \frac{\sum_{t,f}[S(t,f) - \bar{S}_u][T(t-u,f) - \bar{T}]}{\left\{\sum_{t,f}[S(t,f) - \bar{S}_u]^2 \sum_{t,f}[T(t-u,f) - \bar{T}]^2\right\}^{1/2}} \quad -1 \le \gamma(u) \le 1$$

where the sums are over t, f (time, frequency) under the window containing the template T shifted by u time bins; $\bar{S}_u$ is the mean of S under the window containing the shifted template; and $\bar{T}$ is the mean of the template. The normalization of each window of S and T to unit length eliminates the correlation's dependence on acoustic amplitude. The NCC is computed using the fast NCC algorithm, wherein the cross-correlation is computed as a pointwise product in the Fourier domain (Lewis, 1995). Note that the cross-correlation is computed along the time axis only. The NCC is computed using the function *match_template* from the python package *scikit-image* (van der Walt et al., 2014).

All local maxima in the resulting NCC vector are detected, and then filtered based on given criteria. First, correlation peaks below a given magnitude threshold are discarded. Peaks are then filtered based on a minimum time-distance threshold. Detected peaks must be separated by at least half the duration of the template. For this filtering step, all correlation peaks are iterated over in order of descending magnitude. For each peak visited, lower-magnitude peaks within the distance threshold (to the left or right) are discarded from the set. Correlation peaks are detected and filtered using the function *find_peaks* from the python package *SciPy* (Oliphant, 2007). The time coordinates of the correlation peaks satisfying the given criteria are returned as positive detections.

For most template matching analyses in this study, the NCC threshold was chosen to be low (i.e. 0.1). This resulted in a high number of false positives (*fp*'s; we use lowercase notation for template match errors to distinguish them from CNN prediction errors), though the number of false negatives (*fn*'s) was considered negligible.

A parallelized and scalable cloud-based implementation of the template matching algorithm was developed for the ARBIMON Pattern Matching feature, available at (https://arbimon.sieve-analytics.com).

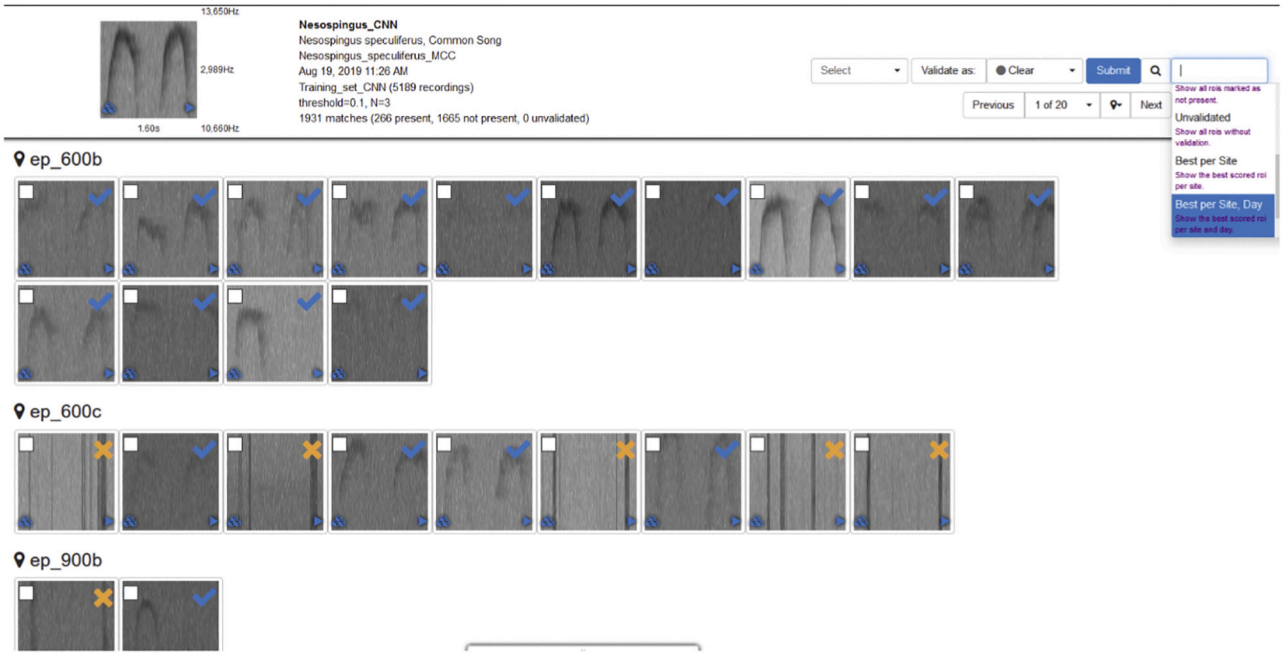


Fig. 3. ARBIMON template match visualizer, allowing for sorting, filtering, and validation of detections, which can be used as CNN training data. In the example screenshot, we show the best matches per day for a *Nesospingus speculiferus* template for three sites.

Input playlists are broken into batches with size determined by the playlist. Each batch is processed with a separate CPU with the ability to launch 1000s of simultaneous CPUs. At typical sampling rates (44.1 or 48 kHz), more than 50,000 1-min recordings (~1 month of audio) can be analyzed for detections in less than five minutes.

2.2.3. Template match validation

The resulting matches can be visually browsed in the ARBIMON platform (Fig. 3). The matches are displayed as time-frequency bounded spectrogram images with the same dimensions as the respective template. Individual detections can also be reviewed with frequency-filtered audio playback, and the source recordings can be easily accessed for audio-visual analysis. The detections can then be validated by checking each image as a true positive (*tp*) or *fp*. Preset queries such as the best *N* matches per recording, site and/or day can accelerate the assessment of the presence or absence of a species at a location.

Altogether, the ARBIMON Visualizer and Pattern Matching features described above allow for highly efficient training data creation (Fig. 3, 4a). Compared to manual inspection and annotation of bounding boxes in spectrograms, many potentially true-positive bounding boxes can be generated automatically in little time, requiring only post-validation.

In total, 512,471 soundscape recordings were searched for call detections using the above three steps. Different recording subsets were searched for different species. This resulted in 86,652 *tp* and 188,908 *fp* annotated time-frequency bounding boxes for 24 species. (Table 1). Note that *fp*'s of a given species *A* that contained the call of a different target species *B* were not re-classified as *tp*'s for *B*. For model training, all *fp*'s were simply used as examples of absence, as detailed in the next section. These true and false detections were used to create the CNN training samples as described below.

2.3. Model training

2.3.1. Training data preprocessing

The CNN model used in this study requires equally sized input images. We chose a time-frequency input window size of 2 s as it is near the mean and median template duration across target call types (Fig. 2). Most call types have a duration below 1 s, and for those above 2 s, important features can still be captured within 2 s. For the case of input frequency bandwidth, we chose to use the entire range of 24 kHz. This was chosen over a smaller, more focused bandwidth for several reasons. Firstly, we approach training as a multi-label classification problem. In other words, for each input audio segment, the model is trained to predict the set of all species present, rather than a single foreground class. This eliminates the need to focus on single target calls in the input. Also, many species' calls highly overlap in frequency (Fig. 2). So, while a focused bandwidth could separate several call types by frequency and avoid their presence together in the same input, many calls would still potentially be present together. A large bandwidth also reduces the number of predictions required to cover a 1-min recording. Furthermore, it increases model generalizability because the optimal bandwidth would likely be specific to the set of target species.

From each template-based detection, we extracted spectrogram images representing two seconds of audio time-centered on the detection and spanning 24 kHz (Fig. 4a). If needed, audio files were resampled to 48 kHz before spectrogram computation. Spectrograms were computed from Hann-windowed 1024-sample (~20 ms) segments of audio data, with 50% segment overlap, and 2048 FFT coefficients per segment. Mel-scaling is a technique commonly applied in acoustic time-frequency analysis. The mel scale refers to a perceptual scale of pitch based on an empirical study of human hearing (Stevens et al., 1937). The conversion from *f* Hertz to *m* "mels" is commonly approximated as:

$$m = 2595\log_{10}\left(1 + \frac{f}{700}\right)$$

Mel-scaling emphasizes lower frequencies while reducing high-frequency resolution, similar to log-scaling the frequency axis. Many of our target call types occupied a low-frequency band (< 6 kHz), which motivated the use of the mel scale. The 1025 frequency bands in our training samples were converted to 128 mel-scaled frequency bands, using the *Librosa* Python package (McFee et al., 2015) (Fig. 4a). Training sample spectrograms were mapped to a color space using the *Librosa* function *specshow*.

2.3.2. Model architecture

We implemented the CNN using the Keras application programming

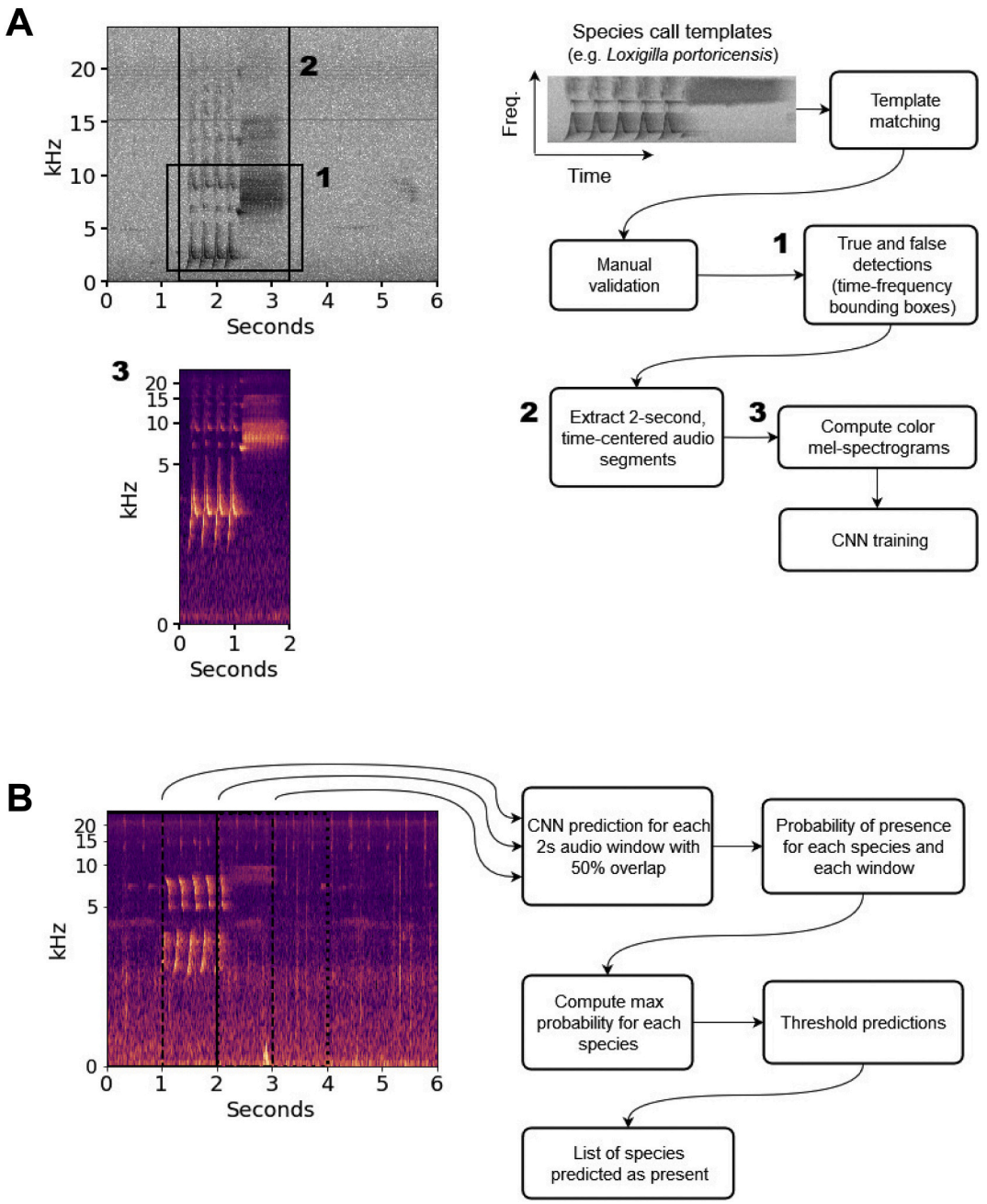


Fig. 4. (Color) Flowcharts for (A) training and (B) prediction portions of the species identification pipeline.

interface to the Tensorflow Python library (Abadi et al., 2016; Chollet, 2019). Code for the CNN training and prediction method described in the paper are available at https://github.com/Sieve-Analytics/arbimon2-cnn. The implementation described here builds on an evaluation of several CNN models for classification of 2-s true and false positive detections of the same species (Zhong et al., 2020). In this case, we choose the best-performing method and apply it to multi-label classification of 1-min soundscape recordings using single-label training data. The training data size and spectrogram parameters are also adjusted, and we did not include a pseudo-labeling step during training. Deep neural networks with initially randomized weights typically require large amounts of training data and time to achieve adequate performance. Improved performance is often obtained from transfer learning, wherein a model already optimized for a similar dataset is re-trained with new data. We used a ResNet50 model, pre-trained on the ImageNet dataset, which contains over one million photo images across 1000 classes (Deng et al., 2009). Although ImageNet does not contain spectrograms, models pre-trained with the dataset learn a variety of image features that have been successfully tuned to spectrogram classification previously (Lasseck, 2019; Florentin et al., 2020). While previous studies have found optimal performance using ensembles of multiple CNN models (Kahl et al., 2017; Lasseck, 2019; Florentin et al., 2020), in the interest of prediction efficiency and reasonable memory requirements, we chose to evaluate a single ResNet50 model.

Our implementation only included the feature extraction layers of ResNet50, excluding the remaining layers used for ImageNet classification, referred to as the network “top”. The network top consists of fully connected (FC) layers for learning a predictive model from the input features. An FC layer consists of a set of nodes, each of which takes a weighted sum of the input's values and passes it through a transfer function. The weights of an FC layer are learnable. In our case, we used two FC layers separated by a drop-out layer. The drop-out layer

causes nodes in the previous FC layer to be probabilistically ignored in each training iteration, such that a different subset of nodes is connected to the final FC layer at each iteration. This emulates training a group of different models and reduces the chance of overfitting. The first FC layer consisted of 512 nodes and used the common "ReLU" activation function, which simply converts negative inputs to 0. The following drop-out layer was assigned a drop-out rate of 0.5, such that each node was ignored with a 50% probability. The final layer consists of 24 nodes, corresponding to the number of target species, and each node was assigned a sigmoid activation function. The sigmoid function S, defined below, maps the input to the range [0,1], and is commonly used for the prediction of binary outcomes. In our case, an independent output score within [0, 1] for each species was desired, to allow for multi-species prediction of presence in audio segments. Thus, our model outputs a vector of 24 scores, representing the predicted probability of presence for each species.

$$S(x) = \frac{1}{1 + e^{-x}}$$

2.3.3. Loss and optimization

A custom training loss function was defined to leverage both true and false detection training data. Typically, for multi-label prediction, the training target vectors consist of 0's and 1's indicating the presence or absence of each class in the input. This assumes fully labeled training data (i.e. all labels are known). In our case, each training sample is only labeled for presence or absence of a single species, based on validation of the associated detection as *tp* or *fp*. Therefore, each target vector consisted of a 1 or 0 at the position of the species determined to be present or absent, and unknown values represented by NaN ("Not a Number") for all other species. The custom loss for class c, given the presence label y_c and predicted score p_c is defined

$$L_c = \begin{cases} -(y_c \log(p_c) + (1 - y_c)\log(1 - p_c)) & y_c \in \{01\} \\ 0 & y_c = NaN \end{cases}$$

The total loss for a sample is simply the loss for the single labeled species. This allows for multi-label learning based on single labels at a time, but requires examples of both presence and absence for each class. This training regime is therefore compatible with true and false-positive detections from other detectors as training data.

The Adam optimization method was used with a learning rate of 1×10^{-4} and decay 1×10^{-7} (Kingma and Ba, 2017). The Adam method allows for an adaptive learning rate during training and is commonly used for deep neural network training. A 10% validation split was used to compare the loss between training samples and unseen samples during training. Based on comparing training and validation loss at the end of each epoch, 5 epochs of training were applied.

2.4. Prediction and evaluation

Performance evaluation was based on the ability to predict the set of species present in each of 1000 randomly sampled 1-min test soundscape recordings from 17 sites in the El Yunque National Forest. As mentioned above, the test sites were selected to cover species-rich habitats, and a variety of elevations and monitoring transects in the forest. To create present-species labels for the test recordings, we used the template matching procedure described in Section 2.2. To do this, for each test recording and each call type, the three highest-correlating template matches were validated as *tp* or *fp*. Due to the low correlation threshold of 0.1, *fn*'s were assumed to be negligible. Therefore, the *tp* detections were used to create present species labels for the test recordings to evaluate model predictions.

The test recordings showed a large class imbalance (Table 2). Frequencies of call presence ranged from 0.3% of recordings (*Melanerpes portoricensis*) to 92.3% of recordings (*Eleutherodactylus coqui*). More than half of the species were in less than 5% of recordings and a quarter were in greater than 20% of the recordings. The number of species in test recordings ranged from 0 to 8, with a mode of 3. Only 1% of test recordings contained no calls, demonstrating the high acoustic activity in the study region.

2.4.1. Sliding window

Prediction was performed using a sliding window approach. Predictions were made for every 2-s time-window of each audio recording with a 1-s shift between windows (50% window overlap) (Fig. 4b). Spectrograms of each audio window were computed in the same way as for training data. For a species and recording, the highest predicted score across audio windows was used to predict presence. We also tested using the average of the two highest scores across windows as the predictive score.

In practice, a threshold τ is applied to the model's predicted scores to make a binary prediction of presence or absence. For a species and recording with predicted score p, we predict presence if $p \geq \tau$, and absence otherwise.

2.4.2. Evaluation metrics

Let $TP \equiv TP(\tau)$ represent number of true-positive CNN predictions (i.e. correct predictions of species presence). Similarly, let *TN* represent number of true negatives (i.e. correct predictions of absence), let *FP* be number of false positives (i.e. incorrect predictions of presence), and let *FN* be number of false negatives (i.e. incorrect predictions of absence). All counts are dependent on the chosen threshold τ.

Performance was quantified by several metrics described below. The precision $P(\tau)$ is the fraction of predictions of presence that are correct.

$$P(\tau) = \frac{TP}{N_P'} = \frac{TP}{TP + FP}$$

where N_P' is the number of predicted presences. Recall $R(\tau)$, also known as sensitivity or true-positive rate, measures the fraction of presences that are correctly identified.

$$R(\tau) = \frac{TP}{N_P} = \frac{TP}{TP + FN}$$

where N_P represents the number of true presences. A precision-recall curve consists of the points in precision-recall space achieved at each possible threshold. Recall is typically the horizontal axis and precision the vertical axis. Based on the precision-recall curve we also measured the average-precision (*AP*) of predictions. The *AP* is defined:

$$AP = \sum_{i=2}^{N} (R(\tau_i) - R(\tau_{i-1}))P(\tau_i)$$

where $(\tau_1, \tau_2, \ldots, \tau_N)$ are the different thresholds to be evaluated, sorted in descending magnitude. Typically, the chosen thresholds $(\tau_1, \tau_2, \ldots, \tau_N)$ are the sorted predicted scores. The *AP* is the weighted sum of precisions at each threshold, using the increase in recall from the previous threshold as the weight. It approximates the integral of, or area under, the precision-recall curve. The *AP* is independent of the chosen threshold τ, so it is commonly used for model comparison. The mean-average-precision (*mAP*) across classes is commonly used in multi-label prediction evaluation.

$$mAP = \frac{1}{N_s} \sum_s AP_s$$

where N_s is the number of target species. The false-positive rate (*FPR*) is the fraction of true absences incorrectly predicted as presences.

$$FPR(\tau) = \frac{FP}{N_A} = \frac{FP}{FP + TN}$$

where N_A represents the number of true absences.

Table 2
Species-specific and summary evaluation scores for a selected prediction threshold of 0.99. The total scores are computed from all predictions across all species and test recordings. The mean scores are computed for each species separately before averaging.

	Presences	Absences	TP	FP	TN	FN	Precision	Recall	FPR
Eleutherodactylus richmondi	123	877	120	2	875	3	0.98	0.98	0.002
Eleutherodactylus coqui	923	77	843	0	77	80	1.00	0.91	0.000
Eleutherodactylus unicolor	474	526	418	1	525	56	1.00	0.88	0.002
Eleutherodactylus portoricensis	255	745	224	1	744	31	1.00	0.88	0.001
Eleutherodactylus locustus	44	956	44	2	954	0	0.96	1.00	0.002
Eleutherodactylus gryllus	113	887	96	1	886	17	0.99	0.85	0.001
Vireo altiloquus	149	851	119	0	851	30	1.00	0.80	0.000
Spindalis portoricensis	72	928	70	4	924	2	0.95	0.97	0.004
Coereba flaveola	213	787	164	0	787	49	1.00	0.77	0.000
Leptodactylus albilabris	44	956	33	0	956	11	1.00	0.75	0.000
Eleutherodactylus antillensis	46	954	34	0	954	12	1.00	0.74	0.000
Eleutherodactylus brittoni	204	796	152	1	795	52	0.99	0.75	0.001
Turdus plumbeus	6	994	4	0	994	2	1.00	0.67	0.000
Patagioenas squamosa	222	778	148	1	777	74	0.99	0.67	0.001
Eleutherodactylus wightmanae	18	982	13	1	981	5	0.93	0.72	0.001
Loxigilla portoricensis	26	974	15	0	974	11	1.00	0.58	0.000
Nesospingus speculiferus	23	977	13	0	977	10	1.00	0.57	0.000
Eleutherodactylus hedricki	53	947	33	2	945	20	0.94	0.62	0.002
Melanerpes portoricensis	3	997	1	0	997	2	1.00	0.33	0.000
Megascops nudipes	14	986	11	6	980	3	0.65	0.79	0.006
Todus mexicanus	17	983	9	6	977	8	0.60	0.53	0.006
Setophaga angelae	41	959	8	0	959	33	1.00	0.20	0.000
Margarops fuscatus	23	977	17	24	953	6	0.41	0.74	0.025
Coccyzus vieilloti	4	996	1	7	989	3	0.13	0.25	0.007
Total	–	–	–	–	–	–	0.98	0.83	0.003
Mean	129.6	870.4	–	–	–	–	0.90	0.71	0.003

2.4.3. Annotation review

Cases where the CNN-predicted score differed from the annotation by 95% or greater were reviewed. For the annotations of presence, 16 cases were found (where the CNN-predicted score was ≤ 0.05) across all species and recordings, of which only 1 was determined to be an annotation error. For the annotations of absence, 294 errors were found (where the CNN-predicted score was ≥ 0.95). Most errors (43%) were caused by the template matching method confusing the target with another call type with higher signal-to-noise-ratio (SNR), causing an *fn*. In 21% of cases, no matches for the target template were found, typically due to low SNR. Other errors were manual annotation errors, mostly based on difficulty validating noisy detections. The number of annotation errors was seen to be positively correlated with difference between annotation and prediction. Furthermore, 78% of all predicted scores for the results shown were below 0.05 or greater than 0.95. Therefore, most annotation errors were assumed to be accounted for. These errors accounted for only 1.2% of annotations. The results presented are based on the revalidated annotation.

3. Results

The average-precision *AP* varied among the species from 1 (*Melanerpes portoricensis*) to 0.28 (*Coccyzus Vielloti*) (Fig. 5). The mean-average-precision across species was 0.893. When computing precision and recall over all species and recordings, the total average-precision (AP_{total}) was 0.975 (Fig. 6, left). This indicates that the species with more frequent calls tended to have higher scores, because each species' contribution to the AP_{total} is proportional to its number of presences. Only 3 of the 24 species had an *AP* below 0.80. Excluding these three species, the mean-average-precision for the rest of the species is 0.955. These results indicate that the model's predicted scores strongly distinguish cases of presence and absence for most species.

Species-specific performance did not have a clear association with training sample size (Table 1, Fig. 5). Many classes with varying sample sizes achieved a similar strong performance. *Melanerpes portoricensis*, *Leptodactylus albilabris*, *Coereba flaveola* each had fewer than 1000 positive training samples and AP $>$ 0.97. Thus, depending on the call type and potentially confounding signals and noise in the environment, strong performance is achievable with several hundreds of call examples.

A high predictive threshold greater than 0.90 was seen to yield an optimal balance of precision and recall (Fig. 6). A balanced precision and recall are desired if false positives and false negatives are equally significant, though if false positives are more costly, precision has greater importance. As the threshold neared 1, the recall began to drop rapidly, particularly for species-average scores. Using the mean of the two highest scores across audio windows to predict presence was seen to provide minor improvements to species-average precision and recall in some cases. However, simply using the highest-scoring window to determine presence theoretically allows for time-localizing the calls, though this was not evaluated. All results presented correspond to a prediction based on the highest score across audio windows for a species and recording.

For a selected prediction threshold of 0.99, the mean precision and recall across species were 0.90 and 0.71, respectively (Table 2). However, the total precision and recall were 0.98 and 0.83, respectively, further demonstrating that model performance is correlated with frequency of species presence. This may be due in part to the slight correlation between species frequency and training sample size, though strong performance was achieved for some species without a large sample size (eg. MEPO, LEAL, COFL). The association between performance and frequency of presence could be further explained by the higher balance between presences and absences. Rarer species with relatively few presences require stronger robustness to noise to achieve a low *FPR* and high precision. Comparing the average-precision scores to the precision and recalls at the chosen threshold of 0.99, we find that some species have a high *AP* but moderate *P* and *R* for the chosen threshold, which suggests that performance could be further improved with species-specific detection thresholds.

Three species with relatively poor scores ($AP < 0.80$) were further investigated: *Margarops fuscatus, Todus mexicanus, Coccyzus vieilloti*. The template call for *Margarops fuscatus* has an approximately two-second, two-syllable structure, which resulted in CNN training samples with call features only near the left and right borders of the image. The training

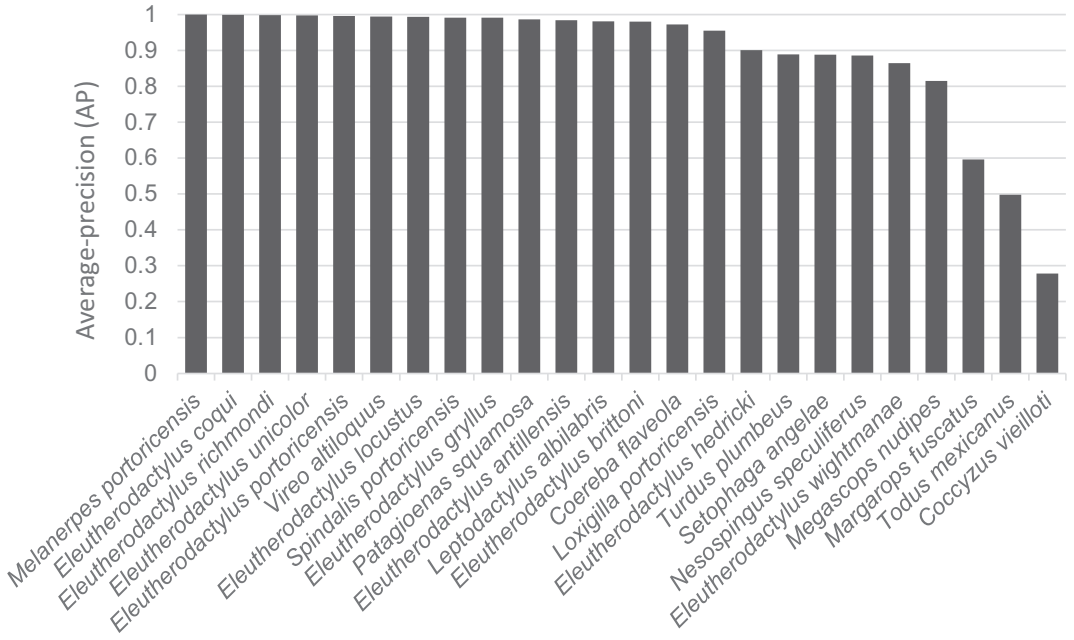


Fig. 5. Average-precision (*AP*) scores for all species.

samples for this species thus had a low signal-to-noise ratio. Similarly, *Todus mexicanus* has a multi-syllable call with a variable number of syllables, of which only one fits in a single CNN input. The multi-syllable structure is a significant feature of the call, though since only a single syllable is seen by the CNN at a time, this presumably impacts the detectability and makes the call more easily confounded with other signals. *Coccyzus vielloti* had clearer training sample features, but the fewest positive training samples of any species (Table 1).

4. Discussion

Soundscape recordings, which capture omnidirectional ambient sound in an area, are widely used for ecological research. While soundscape data is often used to analyze long-term changes in soundscape composition and richness, another interest is to document species-specific site occupancy over time, thus providing quantitative information for species conservation and management decisions. However, due to the non-directional nature of soundscape recordings, species-specific detection methods are often plagued by high false positive rates. Furthermore, multi-species detection models have required high complexity, and thus a high number of training samples to achieve adequate performance.

The work presented here addresses the challenge of acquiring multi-species detections from raw soundscape recordings and demonstrates a high model precision for the study species. Although only bird and frog taxa are considered here, our approach is expected to generalize well to other target signals. No model parameters are specific to the target call types aside from the CNN input width (2 s). The input spans a large frequency bandwidth from 0 to 24 kHz, which can account for many call types. Considering the high variation in the time-frequency extents and characteristics of the target call types in this study, we expect that many vocalizations or other transient signal types (i.e. up to several seconds) within the range of 24 kHz frequency would be appropriate for the pipeline.

The manual effort in training data creation was reduced to template creation and validation of template-based detections in a graphical user interface (Fig. 3). This addresses an important need for more accessible training data from study sites to leverage deep learning for acoustic monitoring. Our evaluation demonstrates that strong classification performance can be achieved using data collected from the study region, without relying on crowd-sourced public datasets. The pipeline thus increases the potential for region-optimized acoustic monitoring systems. Furthermore, the training data collection pipeline could accelerate the collection of data for rare species.

Our training scheme allowed for multi-label learning from single-label training data by defining a custom training loss and including

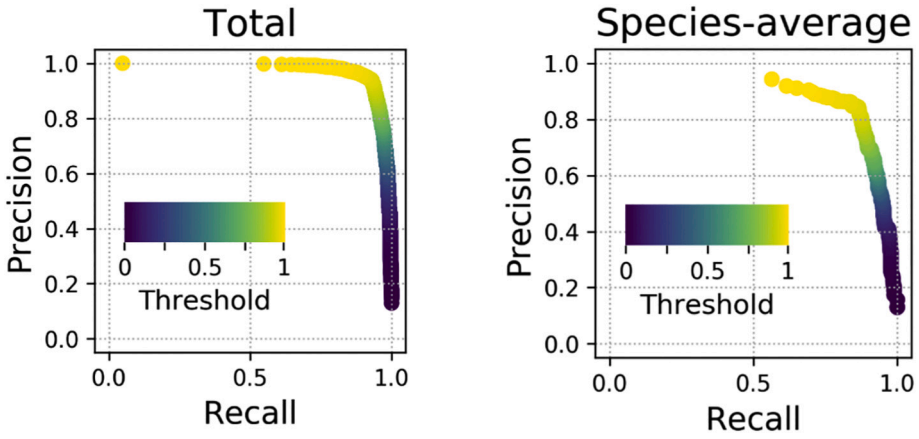


Fig. 6. (Color) (A) Precision-recall curve for all predictions across all species and test recordings. Total precision and recall (i.e. P_{total} and R_{total}) were computed for thresholds from 0 to 1 with an increment of 2.5×10^{-4}. (B) Precision-recall curve where precision and recall are averaged across species at each tested threshold. Mean precision and mean recall were computed for each threshold from 0 to 0.998 with an increment of 0.002. Higher threshold values are not shown in this case because with higher thresholds, all predictions become negative for some species (no presence predicted) and their precision becomes undefined.

both positive and negative examples for each species in training. Instead of using random ambient noise for the absence training samples, we used false template-based detections (*fp*'s). Because *fp*'s were detected based on correlation with the template, this method forces the model to learn to distinguish each call type from similar-looking noise (i.e. unknown) signals as well as from the other target signals. This improved the accuracy compared to tests with a standard approach, in which training was based only on positive examples of each call (Zhong et al., 2020). This is also supported by previous research that reported improved soundscape classification performance when false-positives identified from previous trials were included in the training data for a noise class (Florentin et al., 2020). This method also enables control over the number of positive and negative instances of each class in multi-label learning. Furthermore, this method allows for using the CNN to filter *fp* s from subsequent template-based detections. In general, the CNN can be trained to distinguish positive and negative detections for multiple classes, making it compatible for use in combination with other detectors.

Our mean-average-precision of 0.893 for the 24 species compares favorably to other CNN classification studies. A recent study of CNNs applied to acoustic recognition of six avian species achieved a mean-average-precision of 0.541 (Ruff et al., 2019). The latest BirdCLEF challenge (2019) yielded a maximum *mAP* of 0.407 for soundscape recordings in North America and 0.293 for soundscape recordings from Colombia, though in this case models were trained to identify > 600 species (Kahl et al., 2019). Notably, though, the results in this case were from multi-CNN ensembles, while our results are from a single CNN.

To expand this approach to the broader community, we have identified three important challenges for future research. First, future developments should account for the large variability in the size of target calls (i.e. templates). Introducing recurrent connections in the CNN, or other architecture modifications could potentially reduce the negative effects of window size. Second, previous studies have found data augmentation to significantly improve performance (Kahl et al., 2019). In these cases, training data was mainly based on monodirectional recordings of single species, and data augmentation (i.e. noise addition) apparently helped to emulate the conditions of soundscape recordings. The effect may be reduced for training data collected directly from soundscapes, as in this study. Still, data augmentation may be necessary to increase the training data size for rare species. Thus, future efforts should investigate optimal data augmentation methods for bioacoustic recognition. Third, an important challenge will be to maintain high accuracy while increasing the efficiency of prediction. This will require increasing the prediction speed and decreasing the memory footprint of the model by investigating other network architectures and reducing the number of parameters.

5. Conclusions

The presented pipeline enables training convolutional neural networks for multi-species multi-label classification of soundscape recordings, starting from raw unlabeled recordings. A high-accuracy model for 24 species in the El Yunque National Forest was obtained using training data collected from the study area, without relying on public, labeled bioacoustic datasets. Semi-automated training data collection improves the potential for creating region-specific CNNs for large-scale biodiversity monitoring. We show that single-label true and false-positive detections from a more rudimentary sound detector can be effectively used to train a CNN model for multi-class multi-label sound recognition. False detections, which contain examples of potentially confounding signals from each target, were found to improve performance when incorporated in the training process. Based on our evaluation of the model with 1000, 1-min soundscape recordings, CNNs are a viable solution for automated acoustic monitoring of many species using a single model.

Acknowledgements

Some of the field data were collected for projects funded by U.S. Forest Service (#12F43018C0014) to Sieve Analytics and the National Science Foundation (#1831952) to the University of Puerto Rico. The authors would like to thank Giovany Vega and Michael Haas for their help implementing the research in the ARBIMON platform.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X., 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. http://arxiv.org/abs/1603.04467.

Aide, T.M., Corrada-Bravo, C., Campos-Cerqueira, M., Milan, C., Vega, G., Alvarez, R., 2013. Real-time bioacoustics monitoring and automated species identification. PeerJ 1, e103. https://doi.org/10.7717/peerj.103.

Aloysius, N., Geetha, M., 2017. A review on deep convolutional neural networks. In: 2017 International Conference on Communication and Signal Processing (ICCSP), pp. 0588–0592. https://doi.org/10.1109/ICCSP.2017.8286426.

Chollet, F., 2019. Keras. Online. Available at https://keras.io.

Colonna, J., Peet, T., Ferreira, C., Jorge, A., Gomes, E., Gama, J., 2016. Automatic classification of anuran sounds using convolutional neural networks. In: Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering, pp. 73–78. https://doi.org/10.1145/2948992.2949016.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., 2009. ImageNet: A large-scale hierarchical image database. In: Proceedings of IEEE CVPR 2009.

Florentin, J., Dutoit, T., Verlinden, O., 2020. Detection and identification of European woodpeckers with deep convolutional neural networks. Ecol. Informa. 55, 101023.

Ganchev, T., 2017. Computational Bioacoustics: Biodiversity Monitoring and Assessment. https://doi.org/10.1515/9781614516316.

Goëau, H., Glotin, H., Vellinga, W.P., Planqué, R., Rauber, A., Joly, A., 2015. LifeCLEF Bird Identification Task 2015. Working Notes of CLEF 2015. http://ceur-ws.org/Vol-1391/156-CR.pdf.

Goëau, H., Glotin, H., Vellinga, W.P., Planqué, R., Joly, A., 2016. LifeCLEF bird identification task 2016: The arrival of deep learning. In: Working Notes of CLEF 2016, pp. 440–449. http://ceur-ws.org/Vol-1609/16090440.pdf.

Goëau, H., Glotin, H., Vellinga, W.P., Planqué, R., Joly, A., 2017. LifeCLEF Bird Identification Task 2017. Working Notes of CLEF 2017. http://ceur-ws.org/Vol-1866/invited_paper_8.pdf.

Goëau, H., Kahl, S., Glotin, H., Vellinga, W.P., Planqué, R., Vellinga, W.P., Joly, A., 2018. Overview of BirdCLEF 2018: Monospecies Vs. Soundscape Bird Identification. Working Notes of CLEF 2018. http://ceur-ws.org/Vol-2125/invited_paper_9.pdf.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: Proceedings of CVPR 2016, . http://arxiv.org/abs/1512.03385.

Hill, A.P., Prince, P., Covarrubias, E.P., Doncaster, C.P., Snaddon, J.L., Rogers, A., 2018. AudioMoth: evaluation of a smart open acoustic device for monitoring biodiversity and the environment. Methods Ecol. Evol. 9 (5), 1199–1211. https://doi.org/10.1111/2041-210X.12955.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. CoRR. (vol. abs/1704.04861).

Incze, Á., Jancsó, H.-B., Szilágyi, Z., Farkas, A., Sulyok, C., 2018. Bird sound recognition using a convolutional neural network. In: Proceedings of IEEE 16th Int. Symp. Intell. Syst. Inform. (SISY).

Kahl, S., Willhelm-Stein, T., Hussein, H., Klinck, H., Kowerko, D., Ritter, M., Eibl, M., 2017. Large-Scale Bird Sound Classification Using Convolutional Neural Networks. Working Notes of CLEF 2017. http://ceur-ws.org/Vol-1866/paper_143.pdf.

Kahl, S., Stöter, F.-R., Goëau, H., Glotin, H., Planque, R., Vellinga, W.-P., Joly, A., 2019. Overview of BirdCLEF 2019: Large-Scale Bird Recognition in Soundscapes. Working Notes of CLEF 2019. http://ceur-ws.org/Vol-2380/paper_256.pdf.

Kao, C.-C., Wang, W., Sun, M., Wang, C., 2018. R-CRNN: Region-based convolutional recurrent neural network for audio event detection. In: Proceedings of Interspeech 2018, . http://arxiv.org/abs/1808.06627.

Kingma, D.P., Ba, J., 2017. Adam: A method for stochastic optimization. In: Proceeding of ICLR 2015, . http://arxiv.org/abs/1412.6980.

Koh, C.-Y., Chang, J.-Y., Tai, C.-L., Huang, D.-Y., Hsieh, H.-H., Liu, Y.-W., 2019. Bird Sound Classification Using Convolutional Neural Networks. Working Notes of CLEF 2019. http://ceur-ws.org/Vol-2380/paper_68.pdf.

Lasseck, M., 2019. Bird Species Identification in Soundscapes. Working Notes of CLEF 2019. http://ceur-ws.org/Vol-2380/paper_86.pdf.

Lewis, J., 1995. Fast Normalized Cross-Correlation. Industrial Light and Magic.

McFee, B., Raffel, C., Liang, D., Ellis, D., McVicar, M., Battenberg, E., Nieto, O., 2015. Librosa: Audio and music signal analysis in Python. In: Proceedings of the 14th Python in Science Conference 2015, https://doi.org/10.25080/Majora-7b98e3ed-003.

Oliphant, T., 2007. Python for scientific computing. Comput. Sci. Eng. 9, 10–20. https://doi.org/10.1109/MCSE.2007.58.

Potamitis, I., Ntalampiras, S., Jahn, O., Riede, K., 2014. Automatic bird sound detection in long real-field recordings: applications and tools. Appl. Acoust. 80, 1–9. https://doi.org/10.1016/j.apacoust.2014.01.001.

Priyadarshani, N., Marsland, S., Castro, I., 2018. Automated birdsong recognition in

complex acoustic environments: a review. J. Avian Biol. 49. https://doi.org/10.1111/jav.01447.
Puerto Rico State Wildlife Action Plan, 2015. Ten year review. In: Puerto Rico Department of Natural and Environmental Resources 2015, . http://drna.pr.gov/wp-content/uploads/2015/10/PRSWAP-2015.pdf.
Ruff, Z., Lesmeister, D., Duchac, L., Padmaraju, B., Sullivan, C., 2019. Automated identification of avian vocalizations with deep convolutional neural networks. Remote Sens. Ecol. Conserv. https://doi.org/10.1002/rse2.125.
Sevilla, A., Bessonne, L., Glotin, H., 2017. Audio bird classification with inception-v4 extended with time and time-frequency attention mechanisms. In: Working Notes of CLEF 2017 (Cross Language Evaluation Forum).
Sprengel, E., Jaggi, M., Kilcher, Y., Hofmann, T., 2016. Audio Based Bird Species Identification Using Deep Learning Techniques. Working Notes of CLEF 2016. http://ceur-ws.org/Vol-1609/16090547.pdf.
Stevens, S.S., Volkmann, J., Newman, E.B., 1937. A scale for the measurement of the psychological magnitude pitch. J. Acoust. Soc. Am. 8, 185–190. https://doi.org/10.1121/1.1915893.
Swiston, K., Mennill, D., 2009. Comparison of manual and automated methods for identifying target sounds in audio recordings of pileated, pale-billed, and putative ivory-billed woodpeckers. J. Field Ornithol. 80, 42–50. https://doi.org/10.1111/j.1557-9263.2009.00204.x.
Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions. Proceedings of IEEE Conf. Comput. Vision Pattern Recognition 2015 2015, 1–9.
van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T., scikit-image, contributors., 2014. scikit-image: image processing in Python. PeerJ 2, e453. https://doi.org/10.7717/peerj.453.
Vellinga, W., 2020. Xeno-Canto - Bird Sounds from around the World. Xeno-Canto Foundation for Nature Sounds. Occurrence Dataset. https://doi.org/10.15468/qv0ksn accessed via GBIF.org on 2020-01-22.
Xie, J., Ding, C., Li, W., 2018. Audio-only bird species automated identification method with limited training data based on multi-channel deep convolutional neural networks. arXiv 1803.01107.
Xie, J., Hu, K., Zhu, M., Yu, J., Zhu, Q., 2019. Investigation of different CNN-based models for improved bird sound classification. IEEE Access. 7, 175353–175361. https://doi.org/10.1109/ACCESS.2019.2957572.
Zhang, L., Towsey, M., Xie, J., Zhang, J., Roe, P., 2016. Using multi-label classification for acoustic pattern detection and assisting bird species surveys. Appl. Acoust. 2, 91–98.
Zhong, M., LeBien, J., Campos-Cerquiera, M., Dodhia, R., Ferres, J.L., Velev, J.P., Aide, T.M., 2020. Multispecies bioacoustic classification using transfer learning of deep convolutional neural networks with pseudo-labeling. Appl. Acoust. 166, 107375. https://www.sciencedirect.com/science/article/abs/pii/S0003682X20304795.

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-audio
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]:

```
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	1
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

Rainforest Post Process - Private LB 0.974!

In this notebook we demonstrate a post process for rainforest comp. We will use the output from the following notebook: https://www.kaggle.com/meaninglesslives/rfcx-minimal We will increase it's private LB score from 0.964 to 0.974!

To learn more about this post process, read the discussion here

Post Process Parameters

To use the following post process, load your `submission.csv` file and first try `MODE=1`. Next try `MODE=2`. Next try `MODE=3`. If those three don't increase your LB, then try `MODE=1` with different `FUDGE` values. Try values 0.5, 1, and 3.

In [1]:
```
# USE MODE 1, 2, or 3
MODE = 1

# LOAD SUBMISSION
import pandas as pd, numpy as np
FUDGE = 2.0
FILE = '../input/rfcx-minimal/submission.csv'
df = pd.read_csv(FILE)
for k in range(24):
    df.iloc[:,1+k] -= df.iloc[:,1+k].min()
    df.iloc[:,1+k] /= df.iloc[:,1+k].max()

# CONVERT PROBS TO ODDS, APPLY MULTIPLIER, CONVERT BACK TO PROBS
def scale(probs, factor):
    probs = probs.copy()
    idx = np.where(probs!=1)[0]
    odds = factor * probs[idx] / (1-probs[idx])
    probs[idx] =  odds/(1+odds)
    return probs

# TRAIN AND TEST MEANS
d1 = df.iloc[:,1:].mean().values
d2 = np.array([113,204,44,923,53,41,3,213,44,23,26,149,255,14,123,222,46,6

for k in range(24):
    if MODE==1: d = FUDGE
    if MODE==2: d = d1[k]/(1-d1[k])
    if MODE==3: s = d2[k] / d1[k]
    else: s = (d2[k]/(1-d2[k]))/d
    df.iloc[:,k+1] = scale(df.iloc[:,k+1].values,s)

df.to_csv('submission_with_pp.csv',index=False)
```

Hi to all!!!

I have prepared a notebook that works on both COLAB and KAGGLE!!!

In versions 3 and 4 I show the model I treated on Google COLAB for this notebook https://www.kaggle.com/aikhmelnytskyy/bagging-rainforest

IMPORTANTLY! This notebook didn't work after the changes to kaggle, but thanks to a discussion by Martin Görner and Allohvk (https://www.kaggle.com/c/rfcx-species-audio-detection/discussion/216408), I made the necessary changes in version 4 and now everything works. Here are the changes: from

@tf.function

def _preprocess_img(x, training=False, test=False):

to

@tf.function def _preprocess_img(x, training=False, test=False):

And from

def _specaugment(image):

```
    image = tfa.image.cutout(image, [HEIGHT, xsize[0]], offset=
    [HEIGHT//2, xoff[0]])
    image = tfa.image.cutout(image, [HEIGHT, xsize[1]], offset=
    [HEIGHT//2, xoff[1]])
    image = tfa.image.cutout(image, [ysize[0], WIDTH], offset=
    [yoff[0], WIDTH//2])
    image = tfa.image.cutout(image, [ysize[1], WIDTH], offset=
    [yoff[1], WIDTH//2])
    image = tf.squeeze(image, axis=0)
    return image
```

to

```
    #image = tfa.image.cutout(image, [HEIGHT, xsize[0]], offset=
    [HEIGHT//2, xoff[0]])
    #image = tfa.image.cutout(image, [HEIGHT, xsize[1]], offset=
    [HEIGHT//2, xoff[1]])
    #image = tfa.image.cutout(image, [ysize[0], WIDTH], offset=
    [yoff[0], WIDTH//2])
    #image = tfa.image.cutout(image, [ysize[1], WIDTH], offset=
    [yoff[1], WIDTH//2])
    image = tf.squeeze(image, axis=0)
    return image
```

Version 5 changes as shown in this discussion https://www.kaggle.com/c/rfcx-species-audio-

detection/discussion/218930 (special thanks to the author)
I used these notebooks as a basis: https://www.kaggle.com/mekhdigakhramanian/rfcx-resnet50-tpu https://www.kaggle.com/khoongweihao/resnet34-more-augmentations-mixup-tta-inference

It is important to work with colab you need kaggle.json (https://www.kaggle.com/docs/api)

I also created a folder called Models on my Google Drive and put the kaggle.json file in it.

In [1]:
```
import os

COLAB=False
models_path=''

if not os.path.exists('../input/rfcx-species-audio-detection'):# Let's che
    COLAB=True
    import gc
    from google.colab import drive
    drive.mount('/content/drive')# You must grant COLAB access to your Goo

    #!!!!!!!!!!!!!!!!!!!!!!!!!!!

    GCS_DS_PATH = 'gs://kds-5c677f76ce55440722b2a474a5492faa70847c05a8f5d7
    #This is a path to a dataset that changes over time, so you need to co
    #GCS_DS_PATH = KaggleDatasets (). Get_gcs_path ()
    #print (GCS_DS_PATH)
    models_path='/content/drive/MyDrive/Models/'# I created a folder calle
else:
    from kaggle_datasets import KaggleDatasets
    GCS_DS_PATH = KaggleDatasets().get_gcs_path('rfcx-species-audio-detect
    print (GCS_DS_PATH)
```

```
gs://kds-a7dd8b09d52950714bee1818d843af117accc125d4d289d9afcebfab
```

In [2]:
```
if COLAB:# Prepare the kaggle.json file for use
    from google.colab import files
    if not os.path.exists('/.kaggle/kaggle.json'):
        !mkdir ~/.kaggle
        if not os.path.exists('/content/drive/My Drive/Models/kaggle.json'
            files.upload()
            !cp kaggle.json ~/.kaggle/
        else:
            !cp '/content/drive/My Drive/Models/kaggle.json' ~/.kaggle/
        !chmod 600 ~/.kaggle/kaggle.json
```

In [3]:
```
if COLAB:# force TF to 2.2
    !pip install -q tensorflow~=2.2.0 tensorflow_gcs_config~=2.2.0
    import tensorflow as tf
    import requests
    import os
    resp = requests.post("http://{}:8475/requestversion/{}".format(os.envi
    if resp.status_code != 200:
      print("Failed to switch the TPU to TF {}".format(version))
```

In [4]:
```
if COLAB:    #%tensorflow_version 2.x
    import tensorflow as tf
    print("Tensorflow version " + tf.__version__)

    try:
      tpu = tf.distribute.cluster_resolver.TPUClusterResolver()  # TPU det
      print('Running on TPU ', tpu.cluster_spec().as_dict()['worker'])
    except ValueError:
      raise BaseException('ERROR: Not connected to a TPU runtime; please s

    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
    #tpu_strategy = tf.distribute.TPUStrategy(tpu)
```

In [5]:
```
!pip install -q tensorflow_io
import tensorflow_io as tfio
import tensorflow as tf
import gc
!pip install image-classifiers
!pip install tensorflow_addons==0.10.0
#0.11.2
import tensorflow_addons as tfa
#import tfa as tfa
import numpy as np
from pathlib import Path
import io
import matplotlib.pyplot as plt
!pip install soundfile
import soundfile as sf
import librosa
#!pip install kaggle_datasets

#from kaggle_datasets import KaggleDatasets
from tqdm import tqdm
import pandas as pd
from sklearn.model_selection import StratifiedKFold
import seaborn as sns
from IPython.display import Audio

from classification_models.keras import Classifiers
tf.__version__
```

```
WARNING: You are using pip version 21.0; however, version 21.0.1 is availab
le.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip inst
all --upgrade pip' command.
Collecting image-classifiers
  Downloading image_classifiers-1.0.0-py3-none-any.whl (19 kB)
Collecting keras-applications<=1.0.8,>=1.0.7
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     |████████████████████████████████| 50 kB 1.5 MB/s
Requirement already satisfied: numpy>=1.9.1 in /opt/conda/lib/python3.7/sit
e-packages (from keras-applications<=1.0.8,>=1.0.7->image-classifiers) (1.1
9.5)
Requirement already satisfied: h5py in /opt/conda/lib/python3.7/site-packag
es (from keras-applications<=1.0.8,>=1.0.7->image-classifiers) (2.10.0)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-package
s (from h5py->keras-applications<=1.0.8,>=1.0.7->image-classifiers) (1.15.
0)
```

```
Installing collected packages: keras-applications, image-classifiers
Successfully installed image-classifiers-1.0.0 keras-applications-1.0.8
WARNING: You are using pip version 21.0; however, version 21.0.1 is availab
le.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip inst
all --upgrade pip' command.
Collecting tensorflow_addons==0.10.0
  Downloading tensorflow_addons-0.10.0-cp37-cp37m-manylinux2010_x86_64.whl
(1.0 MB)
     |████████████████████████████████| 1.0 MB 2.9 MB/s
Requirement already satisfied: typeguard>=2.7 in /opt/conda/lib/python3.7/s
ite-packages (from tensorflow_addons==0.10.0) (2.10.0)
Installing collected packages: tensorflow-addons
  Attempting uninstall: tensorflow-addons
    Found existing installation: tensorflow-addons 0.12.0
    Uninstalling tensorflow-addons-0.12.0:
      Successfully uninstalled tensorflow-addons-0.12.0
Successfully installed tensorflow-addons-0.10.0
WARNING: You are using pip version 21.0; however, version 21.0.1 is availab
le.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip inst
/opt/conda/lib/python3.7/site-packages/tensorflow_addons/utils/ensure_tf_in
stall.py:68: UserWarning: Tensorflow Addons supports using Python ops for a
ll Tensorflow versions above or equal to 2.2.0 and strictly below 2.3.0 (ni
ghtly versions are not supported).
 The versions of TensorFlow you are currently using is 2.4.0 and is not sup
ported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration,
either change the TensorFlow version or the TensorFlow Addons's version.
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  UserWarning,
Requirement already satisfied: soundfile in /opt/conda/lib/python3.7/site-p
ackages (0.10.3.post1)
Requirement already satisfied: cffi>=1.0 in /opt/conda/lib/python3.7/site-p
ackages (from soundfile) (1.14.4)
Requirement already satisfied: pycparser in /opt/conda/lib/python3.7/site-p
ackages (from cffi>=1.0->soundfile) (2.20)
WARNING: You are using pip version 21.0; however, version 21.0.1 is availab
le.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip inst
all --upgrade pip' command.
```

Out[5]: '2.4.0'

In [6]:
```
from classification_models.keras import Classifiers
```

In [7]:
```
#SEED = 42
import random
SEED = random.randint(0, 10000)# !!!!
def seed_everything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

seed_everything(SEED)
```

In [8]:
```
# from https://github.com/qubvel/classification_models
ResNet34, preprocess_input = Classifiers.get('resnet34')
```

In [9]:
```
cfg = {
    'parse_params': {
        'cut_time': 10,
    },
    'data_params': {
        'sample_time': 6, # assert 60 % sample_time == 0
        'spec_fmax': 24000.0,
        'spec_fmin': 40.0,
        'spec_mel': 300,
        'mel_power': 2,
        'img_shape': (300, 670)
    },
    'model_params': {
        'batchsize_per_tpu': 8,
        'iteration_per_epoch': 128,
        'epoch': 25,#  1 epoch just for example
        'arch': ResNet34,
        'arch_preprocess': preprocess_input,
        'freeze_to': 0,  # Freeze to backbone.layers[:freeze_to]. If None,
        'loss': {
            'fn': tfa.losses.SigmoidFocalCrossEntropy,
            'params': {},
        },
        'optim': {
            'fn': tfa.optimizers.RectifiedAdam,
            'params': {'lr': 2e-3, 'total_steps': 18*64, 'warmup_proportio
        },
        'mixup': True # False
    }
}
```

In [10]:
```
# detect and init the TPU
if not COLAB:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    print("All devices: ", tf.config.list_logical_devices('TPU'))
    tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
```

```
All devices:  [LogicalDevice(name='/job:worker/replica:0/task:0/device:TPU:
5', device_type='TPU'), LogicalDevice(name='/job:worker/replica:0/task:0/de
vice:TPU:4', device_type='TPU'), LogicalDevice(name='/job:worker/replica:0/
task:0/device:TPU:3', device_type='TPU'), LogicalDevice(name='/job:worker/r
eplica:0/task:0/device:TPU:2', device_type='TPU'), LogicalDevice(name='/jo
b:worker/replica:0/task:0/device:TPU:6', device_type='TPU'), LogicalDevice
(name='/job:worker/replica:0/task:0/device:TPU:7', device_type='TPU'), Logi
calDevice(name='/job:worker/replica:0/task:0/device:TPU:1', device_type='TP
U'), LogicalDevice(name='/job:worker/replica:0/task:0/device:TPU:0', device
_type='TPU')]
```

In [11]:
```
AUTOTUNE = tf.data.experimental.AUTOTUNE


TRAIN_TFREC = GCS_DS_PATH + "/tfrecords/train"
TEST_TFREC = GCS_DS_PATH + "/tfrecords/test"
```

In [12]:
```
CUT = cfg['parse_params']['cut_time']
SR = 48000     # all wave's sample rate may be 48k

TIME = cfg['data_params']['sample_time']

FMAX = cfg['data_params']['spec_fmax']
FMIN = cfg['data_params']['spec_fmin']
N_MEL = cfg['data_params']['spec_mel']

HEIGHT, WIDTH = cfg['data_params']['img_shape']

CLASS_N = 24
```

Explore the tfrecords, Create dataset

In [13]:
```
raw_dataset = tf.data.TFRecordDataset([TRAIN_TFREC + '/00-148.tfrec'])
raw_dataset
```

Out[13]:
```
<TFRecordDatasetV2 shapes: (), types: tf.string>
```

parse tfrecords

In [14]:

```
feature_description = {
    'recording_id': tf.io.FixedLenFeature([], tf.string, default_value='')
    'audio_wav': tf.io.FixedLenFeature([], tf.string, default_value=''),
    'label_info': tf.io.FixedLenFeature([], tf.string, default_value=''),
}
parse_dtype = {
    'audio_wav': tf.float32,
    'recording_id': tf.string,
    'species_id': tf.int32,
    'songtype_id': tf.int32,
    't_min': tf.float32,
    'f_min': tf.float32,
    't_max': tf.float32,
    'f_max':tf.float32,
    'is_tp': tf.int32
}

@tf.function
def _parse_function(example_proto):
    sample = tf.io.parse_single_example(example_proto, feature_description
    wav, _ = tf.audio.decode_wav(sample['audio_wav'], desired_channels=1)
    label_info = tf.strings.split(sample['label_info'], sep='"')[1]
    labels = tf.strings.split(label_info, sep=';')

    @tf.function
    def _cut_audio(label):
        items = tf.strings.split(label, sep=',')
        spid = tf.squeeze(tf.strings.to_number(items[0], tf.int32))
        soid = tf.squeeze(tf.strings.to_number(items[1], tf.int32))
        tmin = tf.squeeze(tf.strings.to_number(items[2]))
        fmin = tf.squeeze(tf.strings.to_number(items[3]))
        tmax = tf.squeeze(tf.strings.to_number(items[4]))
        fmax = tf.squeeze(tf.strings.to_number(items[5]))
        tp = tf.squeeze(tf.strings.to_number(items[6], tf.int32))

        tmax_s = tmax * tf.cast(SR, tf.float32)
        tmin_s = tmin * tf.cast(SR, tf.float32)
        cut_s = tf.cast(CUT * SR, tf.float32)
        all_s = tf.cast(60 * SR, 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
            ), tf.int32
        )
        cut_max = cut_min + CUT * SR

        _sample = {
            'audio_wav': tf.reshape(wav[cut_min:cut_max], [CUT*SR]),
            'recording_id': sample['recording_id'],
            'species_id': spid,
            'songtype_id': soid,
            't_min': tmin - tf.cast(cut_min, tf.float32)/tf.cast(SR, tf.flo
            'f_min': fmin,
            't_max': tmax - tf.cast(cut_min, tf.float32)/tf.cast(SR, tf.flo
            'f_max': fmax,
            'is_tp': tp
        }
        return _sample

    samples = tf.map_fn(_cut_audio, labels, dtype=parse_dtype)
```

In [15]:

```
@tf.function
def _cut_wav(x):
    # random cut in training
    cut_min = tf.random.uniform([], maxval=(CUT-TIME)*SR, dtype=tf.int32)
    cut_max = cut_min + TIME * SR
    cutwave = tf.reshape(x['audio_wav'][cut_min:cut_max], [TIME*SR])
    y = {}
    y.update(x)
    y['audio_wav'] = cutwave
    y['t_min'] = tf.maximum(0.0, x['t_min'] - tf.cast(cut_min, tf.float32)
    y['t_max'] = tf.maximum(0.0, x['t_max'] - tf.cast(cut_min, tf.float32)
    return y

@tf.function
def _cut_wav_val(x):
    # center crop in validation
    cut_min = (CUT-TIME)*SR // 2
    cut_max = cut_min + TIME * SR
    cutwave = tf.reshape(x['audio_wav'][cut_min:cut_max], [TIME*SR])
    y = {}
    y.update(x)
    y['audio_wav'] = cutwave
    y['t_min'] = tf.maximum(0.0, x['t_min'] - tf.cast(cut_min, tf.float32)
    y['t_max'] = tf.maximum(0.0, x['t_max'] - tf.cast(cut_min, tf.float32)
    return y
```

In [16]:

```
@tf.function
def _filtTP(x):
    return x['is_tp'] == 1
```

In [17]:

```
def show_wav(sample, ax):
    wav = sample["audio_wav"].numpy()
    rate = SR
    ax.plot(np.arange(len(wav)) / rate, wav)
    ax.set_title(
        sample["recording_id"].numpy().decode()
        + ("/%d" % sample["species_id"])
        + ("TP" if sample["is_tp"] else "FP"))

    return Audio((wav * 2**15).astype(np.int16), rate=rate)

fig, ax = plt.subplots(figsize=(15, 3))
show_wav(next(iter(parsed_dataset)), ax)
```

Out[17]:

0:00:00 / 12:25:39

003bec244/14TP

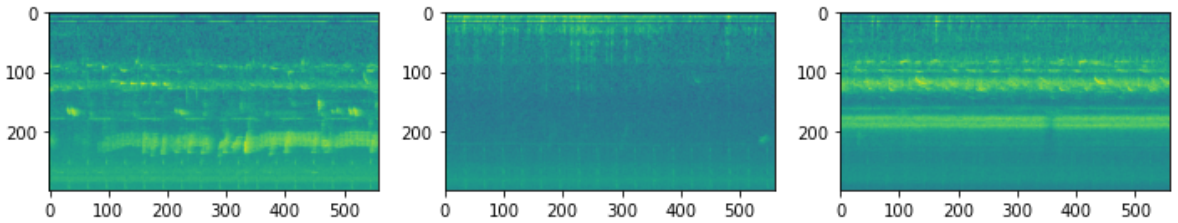
create mel-spectrogram

In [18]:

```
@tf.function
def _wav_to_spec(x):
    mel_power = cfg['data_params']['mel_power']

    stfts = tf.signal.stft(x["audio_wav"], frame_length=2048, frame_step=5
    spectrograms = tf.abs(stfts) ** mel_power

    # Warp the linear scale spectrograms into the mel-scale.
    num_spectrogram_bins = stfts.shape[-1]
    lower_edge_hertz, upper_edge_hertz, num_mel_bins = FMIN, FMAX, N_MEL

    linear_to_mel_weight_matrix = tf.signal.linear_to_mel_weight_matrix(
      num_mel_bins, num_spectrogram_bins, SR, lower_edge_hertz,
      upper_edge_hertz)
    mel_spectrograms = tf.tensordot(
      spectrograms, linear_to_mel_weight_matrix, 1)
    mel_spectrograms.set_shape(spectrograms.shape[:-1].concatenate(
      linear_to_mel_weight_matrix.shape[-1:]))

    # Compute a stabilized log to get log-magnitude mel-scale spectrograms
    log_mel_spectrograms = tf.math.log(mel_spectrograms + 1e-6)

    y = {
        'audio_spec': tf.transpose(log_mel_spectrograms), # (num_mel_bins,
    }
    y.update(x)
    return y

spec_dataset = parsed_dataset.filter(_filtTP).map(_cut_wav).map(_wav_to_spe
```

In [19]:

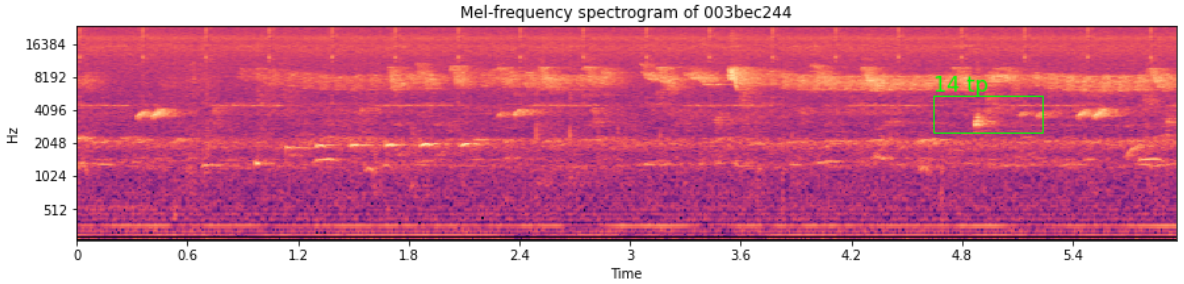
```
plt.figure(figsize=(12,5))
for i, s in enumerate(spec_dataset.take(3)):
    plt.subplot(1,3,i+1)
    plt.imshow(s['audio_spec'])
plt.show()
```



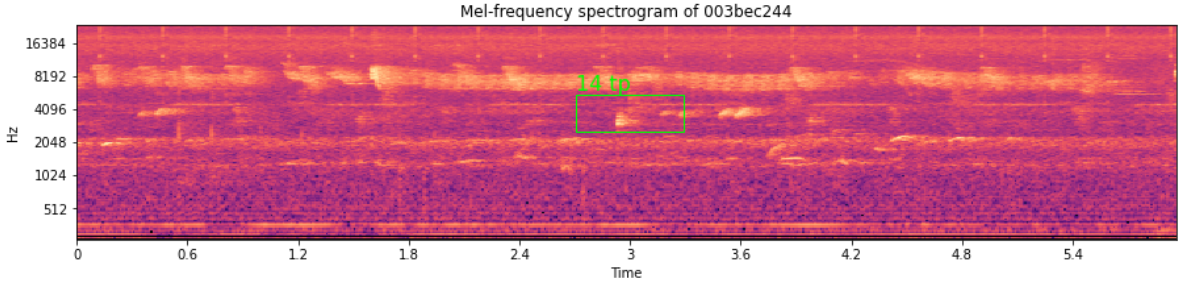
In [20]:
```
import librosa.display
import matplotlib.patches as patches

def show_spectrogram(sample, ax, showlabel=False):
    S_dB = sample["audio_spec"].numpy()
    img = librosa.display.specshow(S_dB, x_axis='time',
                                   y_axis='mel', sr=SR,
                                   fmax=FMAX, fmin=FMIN, ax=ax, cmap='magma')
    ax.set(title=f'Mel-frequency spectrogram of {sample["recording_id"].nu
    sid, fmin, fmax, tmin, tmax, istp = (
            sample["species_id"], sample["f_min"], sample["f_max"], sample
    ec = '#00ff00' if istp == 1 else '#0000ff'
    ax.add_patch(
        patches.Rectangle(xy=(tmin, fmin), width=tmax-tmin, height=fmax-fm
    )

    if showlabel:
        ax.text(tmin, fmax,
        f"{sid.numpy().item()} {'tp' if istp == 1 else 'fp'}",
        horizontalalignment='left', verticalalignment='bottom', color=ec,
```

In [21]:
```
fig, ax = plt.subplots(figsize=(15,3))
show_spectrogram(next(iter(spec_dataset)), ax, showlabel=True)
```

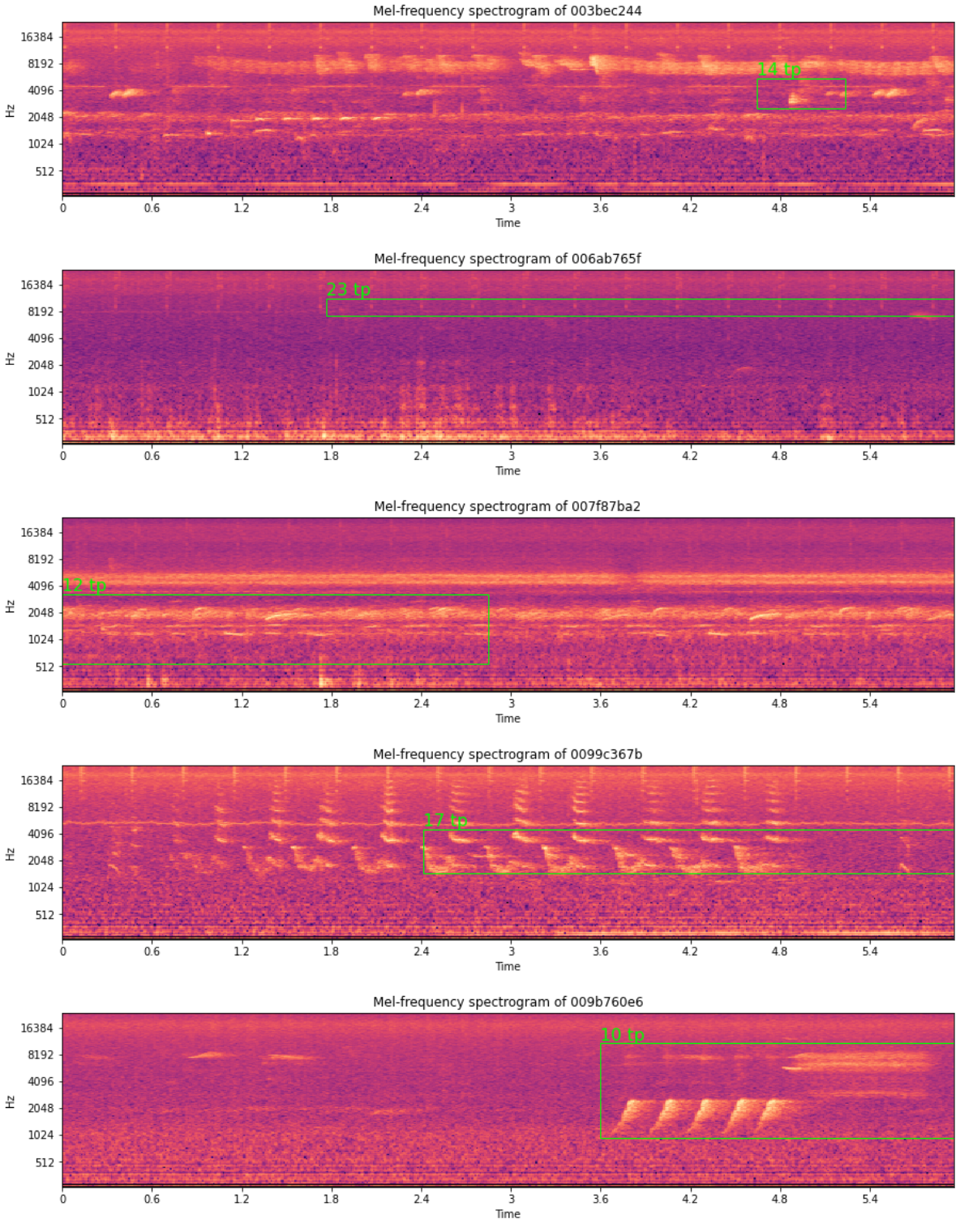
```
/opt/conda/lib/python3.7/site-packages/librosa/display.py:974: MatplotlibDe
precationWarning: The 'basey' parameter of __init__() has been renamed 'bas
e' since Matplotlib 3.3; support for the old name will be dropped two minor
releases later.
  scaler(mode, **kwargs)
/opt/conda/lib/python3.7/site-packages/librosa/display.py:974: MatplotlibDe
precationWarning: The 'linthreshy' parameter of __init__() has been renamed
'linthresh' since Matplotlib 3.3; support for the old name will be dropped
two minor releases later.
  scaler(mode, **kwargs)
```



In [22]:
```
# in validation, annotations will come to the center
fig, ax = plt.subplots(figsize=(15,3))
show_spectrogram(next(iter(parsed_dataset.filter(_filtTP).map(_cut_wav_val
```



In [23]:
```
for sample in spec_dataset.take(5):
    fig, ax = plt.subplots(figsize=(15,3))
    show_spectrogram(sample, ax, showlabel=True)
```



create labels

In [24]:

```
@tf.function
def _create_annot(x):
    targ = tf.one_hot(x["species_id"], CLASS_N, on_value=x["is_tp"], off_v

    return {
        'input': x["audio_spec"],
        'target': tf.cast(targ, tf.float32)
    }

annot_dataset = spec_dataset.map(_create_annot)
```

proprocessing and data augmentation

In training, I use

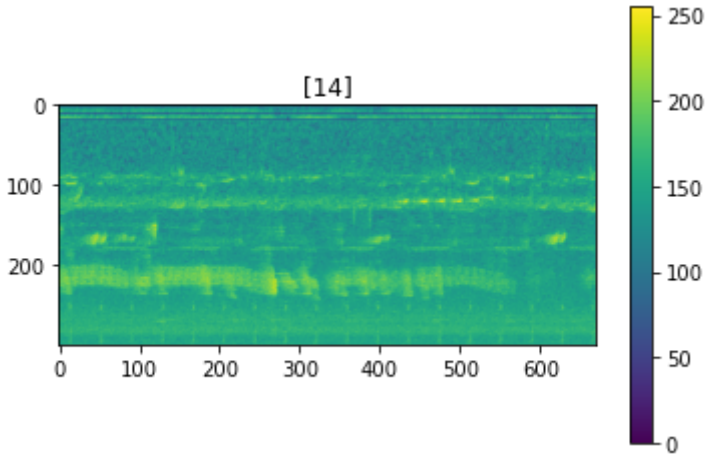
- gaussian noise
- random brightness
- specaugment

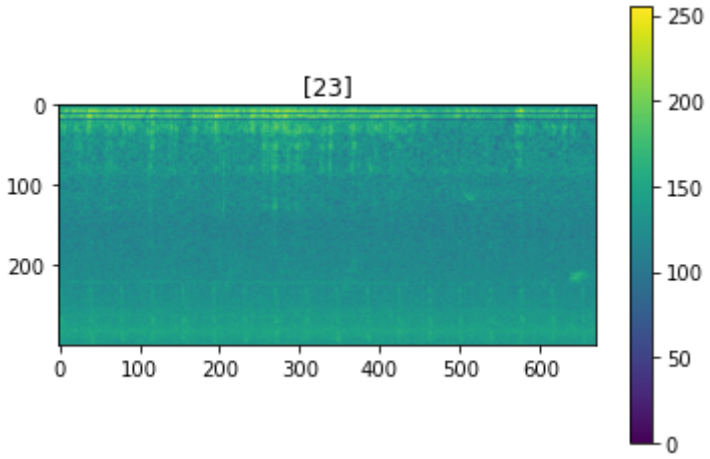
In [25]:

```
#@tf.function
def _preprocess_img(x, training=False, test=False):
    image = tf.expand_dims(x, axis=-1)
    image = tf.image.resize(image, [HEIGHT, WIDTH])
    image = tf.image.per_image_standardization(image)

    @tf.function
    def _specaugment(image):
        ERASE_TIME = 50
        ERASE_MEL = 16
        image = tf.squeeze(image, axis=2)
        image = tfio.experimental.audio.time_mask(image, param=ERASE_TIME)
        image = tfio.experimental.audio.freq_mask(image, param=ERASE_MEL)
        image = tf.expand_dims(image, axis=2)
        return image

    if training:
        # gaussian
        gau = tf.keras.layers.GaussianNoise(0.3)
        image = tf.cond(tf.random.uniform([]) < 0.5, lambda: gau(image, tra
        # brightness
        image = tf.image.random_brightness(image, 0.2)
        # random left right flip (NEW)
        image = tf.image.random_flip_left_right(image)
        # specaugment
        image = tf.cond(tf.random.uniform([]) < 0.5, lambda: _specaugment(i

    if test:
        # specaugment
        image = tf.cond(tf.random.uniform([]) < 0.5, lambda: _specaugment(i

    image = (image - tf.reduce_min(image)) / (tf.reduce_max(image) - tf.red
    image = tf.image.grayscale_to_rgb(image)
    image = cfg['model_params']['arch_preprocess'](image)

    return image

@tf.function
def _preprocess(x):
    image = _preprocess_img(x['input'], training=True, test=False)
    return (image, x["target"])

@tf.function
def _preprocess_val(x):
    image = _preprocess_img(x['input'], training=False, test=False)
    return (image, x["target"])

@tf.function
def _preprocess_test(x):
    image = _preprocess_img(x['audio_spec'], training=False, test=True)
    return (image, x["recording_id"])
```

In [26]:

```
for inp, targ in annot_dataset.map(_preprocess).take(2):
    plt.imshow(inp.numpy()[:,:,0])
    t = targ.numpy()
    if t.sum() == 0:
        plt.title(f'FP')
    else:
        plt.title(f'{t.nonzero()[0]}')
    plt.colorbar()
    plt.show()
```





Model

In [27]:

```
from tensorflow.keras.layers import *
from tensorflow.keras import losses, models, optimizers
from tensorflow.keras.optimizers import Adam
def create_model():
    #with strategy.scope():
    #backbone = cfg['model_params']['arch'](include_top=False, weights='im

    def Classifier(shape_):

        backbone = cfg['model_params']['arch']((shape_), include_top=False



        def cbr(x, out_layer, kernel, stride, dilation):
            x = Conv2D(out_layer, kernel_size=kernel, dilation_rate=dilati
            x = BatchNormalization()(x)
            x = Activation("relu")(x)
            return x

        def wave_block(x, filters, kernel_size, n):
            dilation_rates = [2**i for i in range(n)]
            x = Conv2D(filters = filters,
                       kernel_size = 1,
                       padding = 'same')(x)
            res_x = x
            for dilation_rate in dilation_rates:
                tanh_out = Conv2D(filters = filters,
                                  kernel_size = kernel_size,
                                  padding = 'same',
                                  activation = 'tanh',
                                  dilation_rate = dilation_rate)(x)
                sigm_out = Conv2D(filters = filters,
                                  kernel_size = kernel_size,
                                  padding = 'same',
                                  activation = 'sigmoid',
                                  dilation_rate = dilation_rate)(x)
                x = Multiply()([tanh_out, sigm_out])
                x = Conv2D(filters = filters,
                           kernel_size = 1,
                           padding = 'same')(x)
                res_x = Add()([res_x, x])
            return res_x


        #out1
        def wavenet(layer):

          x = cbr(layer, 192, 7, 1, 1)
          x = BatchNormalization()(x)
          x = wave_block(x, 192, 3, 1)
          x = cbr(x, 96, 7, 1, 1)
          x = BatchNormalization()(x)
          x = wave_block(x, 96, 3, 1)
          x = cbr(x, 48, 5, 1, 1)
          x = BatchNormalization()(x)
          x = wave_block(x, 48, 3, 1)
          return x

        def wavenet1(layer):

          x = cbr(layer, 4, 7, 1, 1)
          x = BatchNormalization()(x)
```

```
        x0 = backbone#model
        print('1')
        #backbone.summary()
        x1 = tf.keras.layers.GlobalAveragePooling2D()(x0.layers[-1].output
        #x2 = tf.keras.layers.GlobalAveragePooling2D()(x0.layers[-3].outpu
        x3 = tf.keras.layers.GlobalAveragePooling2D()(x0.layers[-7].output
        #x4 = tf.keras.layers.GlobalAveragePooling2D()(x0.layers[-12].outp
        x5 = tf.keras.layers.GlobalAveragePooling2D()(x0.layers[-18].outpu
        print('2')
        x1=wavenet(x0.layers[-1].output)
        x3=wavenet(x0.layers[-7].output)
        x5=wavenet(x0.layers[-18].output)

        x1 = tf.keras.layers.GlobalAveragePooling2D()(x1)
        x3 = tf.keras.layers.GlobalAveragePooling2D()(x3)
        x5 = tf.keras.layers.GlobalAveragePooling2D()(x5)



        print('4')
        #x =  tf.concat([x1,x2,x3,x4,x5],axis = 1)

        x =  tf.concat([x1,x3,x5],axis = 1)

        x = tf.keras.layers.Dropout(0.7)(x)
        x = tf.keras.layers.Dense(192)(x)
        #x =  tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Dropout(0.4)(x)
        #x =  margin([x , label])

        output = tf.keras.layers.Softmax(dtype='float32')(x)
        output =tf.keras.layers.Dense(CLASS_N)(x)
        print('5')
        model = tf.keras.models.Model(inputs = x0.input, outputs = output)
        #model.compile(optimizer=optimizer, loss=loss_fn, metrics=[LWLRAP(



        return model
    return Classifier([HEIGHT,WIDTH,3])


model = create_model()
model.summary()
```

```
Downloading data from https://github.com/qubvel/classification_models/relea
ses/download/0.0.1/resnet34_imagenet_1000_no_top.h5
85524480/85521592 [==============================] - 1s 0us/step
1
2
4
5
Model: "model_1"
__________________________________________________________________________
```

```
__________________________
Layer (type)                 Output Shape         Param #     Connected
to
==========================================================================
======================
data (InputLayer)            [(None, 300, 670, 3) 0
__________________________________________________________________________
__________________________
bn_data (BatchNormalization) (None, 300, 670, 3)  9           data[0][0]
__________________________________________________________________________
__________________________
zero_padding2d (ZeroPadding2D) (None, 306, 676, 3)  0         bn_data
[0][0]
__________________________________________________________________________
__________________________
conv0 (Conv2D)               (None, 150, 335, 64) 9408        zero_paddi
ng2d[0][0]
__________________________________________________________________________
__________________________
bn0 (BatchNormalization)     (None, 150, 335, 64) 256         conv0
[0][0]
__________________________________________________________________________
__________________________
relu0 (Activation)           (None, 150, 335, 64) 0           bn0[0][0]
__________________________________________________________________________
__________________________
zero_padding2d_1 (ZeroPadding2D (None, 152, 337, 64) 0        relu0
[0][0]
__________________________________________________________________________
__________________________
pooling0 (MaxPooling2D)      (None, 75, 168, 64)  0           zero_paddi
ng2d_1[0][0]
__________________________________________________________________________
__________________________
stage1_unit1_bn1 (BatchNormaliz (None, 75, 168, 64)  256      pooling0
[0][0]
__________________________________________________________________________
__________________________
stage1_unit1_relu1 (Activation) (None, 75, 168, 64)  0        stage1_uni
t1_bn1[0][0]
__________________________________________________________________________
__________________________
zero_padding2d_2 (ZeroPadding2D (None, 77, 170, 64)  0        stage1_uni
t1_relu1[0][0]
__________________________________________________________________________
__________________________
stage1_unit1_conv1 (Conv2D)  (None, 75, 168, 64)  36864       zero_paddi
ng2d_2[0][0]
__________________________________________________________________________
__________________________
stage1_unit1_bn2 (BatchNormaliz (None, 75, 168, 64)  256      stage1_uni
t1_conv1[0][0]
__________________________________________________________________________
__________________________
stage1_unit1_relu2 (Activation) (None, 75, 168, 64)  0        stage1_uni
t1_bn2[0][0]
__________________________________________________________________________
__________________________
zero_padding2d_3 (ZeroPadding2D (None, 77, 170, 64)  0        stage1_uni
t1_relu2[0][0]
__________________________________________________________________________
__________________________
stage1_unit1_conv2 (Conv2D)  (None, 75, 168, 64)  36864       zero_paddi
ng2d_3[0][0]
```

```
__________________________________________________________________________________________________
____________________________
stage1_unit1_sc (Conv2D)        (None, 75, 168, 64)  4096        stage1_uni
t1_relu1[0][0]
__________________________________________________________________________________________________
____________________________
add (Add)                       (None, 75, 168, 64)  0           stage1_uni
t1_conv2[0][0]
                                                                 stage1_uni
t1_sc[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit2_bn1 (BatchNormaliz (None, 75, 168, 64)  256         add[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit2_relu1 (Activation) (None, 75, 168, 64)  0           stage1_uni
t2_bn1[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_4 (ZeroPadding2D (None, 77, 170, 64)  0           stage1_uni
t2_relu1[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit2_conv1 (Conv2D)     (None, 75, 168, 64)  36864       zero_paddi
ng2d_4[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit2_bn2 (BatchNormaliz (None, 75, 168, 64)  256         stage1_uni
t2_conv1[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit2_relu2 (Activation) (None, 75, 168, 64)  0           stage1_uni
t2_bn2[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_5 (ZeroPadding2D (None, 77, 170, 64)  0           stage1_uni
t2_relu2[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit2_conv2 (Conv2D)     (None, 75, 168, 64)  36864       zero_paddi
ng2d_5[0][0]
__________________________________________________________________________________________________
____________________________
add_1 (Add)                     (None, 75, 168, 64)  0           stage1_uni
t2_conv2[0][0]
                                                                 add[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit3_bn1 (BatchNormaliz (None, 75, 168, 64)  256         add_1
[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit3_relu1 (Activation) (None, 75, 168, 64)  0           stage1_uni
t3_bn1[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_6 (ZeroPadding2D (None, 77, 170, 64)  0           stage1_uni
t3_relu1[0][0]
__________________________________________________________________________________________________
____________________________
stage1_unit3_conv1 (Conv2D)     (None, 75, 168, 64)  36864       zero_paddi
ng2d_6[0][0]
__________________________________________________________________________________________________
____________________________
```

```
stage1_unit3_bn2 (BatchNormaliz (None, 75, 168, 64)  256         stage1_uni
t3_conv1[0][0]
__________________________________________________________________________
_____________________
stage1_unit3_relu2 (Activation) (None, 75, 168, 64)  0           stage1_uni
t3_bn2[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_7 (ZeroPadding2D (None, 77, 170, 64)  0           stage1_uni
t3_relu2[0][0]
__________________________________________________________________________
_____________________
stage1_unit3_conv2 (Conv2D)     (None, 75, 168, 64)  36864       zero_paddi
ng2d_7[0][0]
__________________________________________________________________________
_____________________
add_2 (Add)                     (None, 75, 168, 64)  0           stage1_uni
t3_conv2[0][0]
                                                                 add_1
[0][0]
__________________________________________________________________________
_____________________
stage2_unit1_bn1 (BatchNormaliz (None, 75, 168, 64)  256         add_2
[0][0]
__________________________________________________________________________
_____________________
stage2_unit1_relu1 (Activation) (None, 75, 168, 64)  0           stage2_uni
t1_bn1[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_8 (ZeroPadding2D (None, 77, 170, 64)  0           stage2_uni
t1_relu1[0][0]
__________________________________________________________________________
_____________________
stage2_unit1_conv1 (Conv2D)     (None, 38, 84, 128)  73728       zero_paddi
ng2d_8[0][0]
__________________________________________________________________________
_____________________
stage2_unit1_bn2 (BatchNormaliz (None, 38, 84, 128)  512         stage2_uni
t1_conv1[0][0]
__________________________________________________________________________
_____________________
stage2_unit1_relu2 (Activation) (None, 38, 84, 128)  0           stage2_uni
t1_bn2[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_9 (ZeroPadding2D (None, 40, 86, 128)  0           stage2_uni
t1_relu2[0][0]
__________________________________________________________________________
_____________________
stage2_unit1_conv2 (Conv2D)     (None, 38, 84, 128)  147456      zero_paddi
ng2d_9[0][0]
__________________________________________________________________________
_____________________
stage2_unit1_sc (Conv2D)        (None, 38, 84, 128)  8192        stage2_uni
t1_relu1[0][0]
__________________________________________________________________________
_____________________
add_3 (Add)                     (None, 38, 84, 128)  0           stage2_uni
t1_conv2[0][0]
                                                                 stage2_uni
t1_sc[0][0]
__________________________________________________________________________
_____________________
```

```
stage2_unit2_bn1 (BatchNormaliz (None, 38, 84, 128)  512         add_3
[0][0]
__________________________________________________________________________________________________
stage2_unit2_relu1 (Activation) (None, 38, 84, 128)  0           stage2_uni
t2_bn1[0][0]
__________________________________________________________________________________________________
zero_padding2d_10 (ZeroPadding2 (None, 40, 86, 128)  0           stage2_uni
t2_relu1[0][0]
__________________________________________________________________________________________________
stage2_unit2_conv1 (Conv2D)     (None, 38, 84, 128)  147456      zero_paddi
ng2d_10[0][0]
__________________________________________________________________________________________________
stage2_unit2_bn2 (BatchNormaliz (None, 38, 84, 128)  512         stage2_uni
t2_conv1[0][0]
__________________________________________________________________________________________________
stage2_unit2_relu2 (Activation) (None, 38, 84, 128)  0           stage2_uni
t2_bn2[0][0]
__________________________________________________________________________________________________
zero_padding2d_11 (ZeroPadding2 (None, 40, 86, 128)  0           stage2_uni
t2_relu2[0][0]
__________________________________________________________________________________________________
stage2_unit2_conv2 (Conv2D)     (None, 38, 84, 128)  147456      zero_paddi
ng2d_11[0][0]
__________________________________________________________________________________________________
add_4 (Add)                     (None, 38, 84, 128)  0           stage2_uni
t2_conv2[0][0]
                                                                 add_3
[0][0]
__________________________________________________________________________________________________
stage2_unit3_bn1 (BatchNormaliz (None, 38, 84, 128)  512         add_4
[0][0]
__________________________________________________________________________________________________
stage2_unit3_relu1 (Activation) (None, 38, 84, 128)  0           stage2_uni
t3_bn1[0][0]
__________________________________________________________________________________________________
zero_padding2d_12 (ZeroPadding2 (None, 40, 86, 128)  0           stage2_uni
t3_relu1[0][0]
__________________________________________________________________________________________________
stage2_unit3_conv1 (Conv2D)     (None, 38, 84, 128)  147456      zero_paddi
ng2d_12[0][0]
__________________________________________________________________________________________________
stage2_unit3_bn2 (BatchNormaliz (None, 38, 84, 128)  512         stage2_uni
t3_conv1[0][0]
__________________________________________________________________________________________________
stage2_unit3_relu2 (Activation) (None, 38, 84, 128)  0           stage2_uni
t3_bn2[0][0]
__________________________________________________________________________________________________
zero_padding2d_13 (ZeroPadding2 (None, 40, 86, 128)  0           stage2_uni
t3_relu2[0][0]
```

```
__________________________________________________________________________________________________
____________________________
stage2_unit3_conv2 (Conv2D)     (None, 38, 84, 128)  147456      zero_paddi
ng2d_13[0][0]
__________________________________________________________________________________________________
____________________________
add_5 (Add)                     (None, 38, 84, 128)  0           stage2_uni
t3_conv2[0][0]
                                                                 add_4
[0][0]
__________________________________________________________________________________________________
____________________________
stage2_unit4_bn1 (BatchNormaliz (None, 38, 84, 128)  512         add_5
[0][0]
__________________________________________________________________________________________________
____________________________
stage2_unit4_relu1 (Activation) (None, 38, 84, 128)  0           stage2_uni
t4_bn1[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_14 (ZeroPadding2 (None, 40, 86, 128)  0           stage2_uni
t4_relu1[0][0]
__________________________________________________________________________________________________
____________________________
stage2_unit4_conv1 (Conv2D)     (None, 38, 84, 128)  147456      zero_paddi
ng2d_14[0][0]
__________________________________________________________________________________________________
____________________________
stage2_unit4_bn2 (BatchNormaliz (None, 38, 84, 128)  512         stage2_uni
t4_conv1[0][0]
__________________________________________________________________________________________________
____________________________
stage2_unit4_relu2 (Activation) (None, 38, 84, 128)  0           stage2_uni
t4_bn2[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_15 (ZeroPadding2 (None, 40, 86, 128)  0           stage2_uni
t4_relu2[0][0]
__________________________________________________________________________________________________
____________________________
stage2_unit4_conv2 (Conv2D)     (None, 38, 84, 128)  147456      zero_paddi
ng2d_15[0][0]
__________________________________________________________________________________________________
____________________________
add_6 (Add)                     (None, 38, 84, 128)  0           stage2_uni
t4_conv2[0][0]
                                                                 add_5
[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit1_bn1 (BatchNormaliz (None, 38, 84, 128)  512         add_6
[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit1_relu1 (Activation) (None, 38, 84, 128)  0           stage3_uni
t1_bn1[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_16 (ZeroPadding2 (None, 40, 86, 128)  0           stage3_uni
t1_relu1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit1_conv1 (Conv2D)     (None, 19, 42, 256)  294912      zero_paddi
ng2d_16[0][0]
```

```
__________________________________________________________________________________________________
____________________________
stage3_unit1_bn2 (BatchNormaliz (None, 19, 42, 256)  1024        stage3_uni
t1_conv1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit1_relu2 (Activation) (None, 19, 42, 256)  0           stage3_uni
t1_bn2[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_17 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t1_relu2[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit1_conv2 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_17[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit1_sc (Conv2D)        (None, 19, 42, 256)  32768       stage3_uni
t1_relu1[0][0]
__________________________________________________________________________________________________
____________________________
add_7 (Add)                     (None, 19, 42, 256)  0           stage3_uni
t1_conv2[0][0]
                                                                 stage3_uni
t1_sc[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit2_bn1 (BatchNormaliz (None, 19, 42, 256)  1024        add_7
[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit2_relu1 (Activation) (None, 19, 42, 256)  0           stage3_uni
t2_bn1[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_18 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t2_relu1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit2_conv1 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_18[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit2_bn2 (BatchNormaliz (None, 19, 42, 256)  1024        stage3_uni
t2_conv1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit2_relu2 (Activation) (None, 19, 42, 256)  0           stage3_uni
t2_bn2[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_19 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t2_relu2[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit2_conv2 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_19[0][0]
__________________________________________________________________________________________________
____________________________
add_8 (Add)                     (None, 19, 42, 256)  0           stage3_uni
t2_conv2[0][0]
                                                                 add_7
[0][0]
```

```
__________________________________________________________________________________________________
____________________________
stage3_unit3_bn1 (BatchNormaliz (None, 19, 42, 256)  1024        add_8
[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit3_relu1 (Activation) (None, 19, 42, 256)  0           stage3_uni
t3_bn1[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_20 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t3_relu1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit3_conv1 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_20[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit3_bn2 (BatchNormaliz (None, 19, 42, 256)  1024        stage3_uni
t3_conv1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit3_relu2 (Activation) (None, 19, 42, 256)  0           stage3_uni
t3_bn2[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_21 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t3_relu2[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit3_conv2 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_21[0][0]
__________________________________________________________________________________________________
____________________________
add_9 (Add)                     (None, 19, 42, 256)  0           stage3_uni
t3_conv2[0][0]
                                                                 add_8
[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit4_bn1 (BatchNormaliz (None, 19, 42, 256)  1024        add_9
[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit4_relu1 (Activation) (None, 19, 42, 256)  0           stage3_uni
t4_bn1[0][0]
__________________________________________________________________________________________________
____________________________
zero_padding2d_22 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t4_relu1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit4_conv1 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_22[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit4_bn2 (BatchNormaliz (None, 19, 42, 256)  1024        stage3_uni
t4_conv1[0][0]
__________________________________________________________________________________________________
____________________________
stage3_unit4_relu2 (Activation) (None, 19, 42, 256)  0           stage3_uni
t4_bn2[0][0]
__________________________________________________________________________________________________
____________________________
```

```
zero_padding2d_23 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t4_relu2[0][0]
__________________________________________________________________________
_____________________
stage3_unit4_conv2 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_23[0][0]
__________________________________________________________________________
_____________________
add_10 (Add)                    (None, 19, 42, 256)  0           stage3_uni
t4_conv2[0][0]
                                                                 add_9
[0][0]
__________________________________________________________________________
_____________________
stage3_unit5_bn1 (BatchNormaliz (None, 19, 42, 256)  1024        add_10
[0][0]
__________________________________________________________________________
_____________________
stage3_unit5_relu1 (Activation) (None, 19, 42, 256)  0           stage3_uni
t5_bn1[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_24 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t5_relu1[0][0]
__________________________________________________________________________
_____________________
stage3_unit5_conv1 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_24[0][0]
__________________________________________________________________________
_____________________
stage3_unit5_bn2 (BatchNormaliz (None, 19, 42, 256)  1024        stage3_uni
t5_conv1[0][0]
__________________________________________________________________________
_____________________
stage3_unit5_relu2 (Activation) (None, 19, 42, 256)  0           stage3_uni
t5_bn2[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_25 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t5_relu2[0][0]
__________________________________________________________________________
_____________________
stage3_unit5_conv2 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_25[0][0]
__________________________________________________________________________
_____________________
add_11 (Add)                    (None, 19, 42, 256)  0           stage3_uni
t5_conv2[0][0]
                                                                 add_10
[0][0]
__________________________________________________________________________
_____________________
stage3_unit6_bn1 (BatchNormaliz (None, 19, 42, 256)  1024        add_11
[0][0]
__________________________________________________________________________
_____________________
stage3_unit6_relu1 (Activation) (None, 19, 42, 256)  0           stage3_uni
t6_bn1[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_26 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t6_relu1[0][0]
__________________________________________________________________________
_____________________
```

```
stage3_unit6_conv1 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_26[0][0]
__________________________________________________________________________
_____________________
stage3_unit6_bn2 (BatchNormaliz (None, 19, 42, 256)  1024        stage3_uni
t6_conv1[0][0]
__________________________________________________________________________
_____________________
stage3_unit6_relu2 (Activation) (None, 19, 42, 256)  0           stage3_uni
t6_bn2[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_27 (ZeroPadding2 (None, 21, 44, 256)  0           stage3_uni
t6_relu2[0][0]
__________________________________________________________________________
_____________________
stage3_unit6_conv2 (Conv2D)     (None, 19, 42, 256)  589824      zero_paddi
ng2d_27[0][0]
__________________________________________________________________________
_____________________
add_12 (Add)                    (None, 19, 42, 256)  0           stage3_uni
t6_conv2[0][0]
                                                                 add_11
[0][0]
__________________________________________________________________________
_____________________
stage4_unit1_bn1 (BatchNormaliz (None, 19, 42, 256)  1024        add_12
[0][0]
__________________________________________________________________________
_____________________
stage4_unit1_relu1 (Activation) (None, 19, 42, 256)  0           stage4_uni
t1_bn1[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_28 (ZeroPadding2 (None, 21, 44, 256)  0           stage4_uni
t1_relu1[0][0]
__________________________________________________________________________
_____________________
stage4_unit1_conv1 (Conv2D)     (None, 10, 21, 512)  1179648     zero_paddi
ng2d_28[0][0]
__________________________________________________________________________
_____________________
stage4_unit1_bn2 (BatchNormaliz (None, 10, 21, 512)  2048        stage4_uni
t1_conv1[0][0]
__________________________________________________________________________
_____________________
stage4_unit1_relu2 (Activation) (None, 10, 21, 512)  0           stage4_uni
t1_bn2[0][0]
__________________________________________________________________________
_____________________
zero_padding2d_29 (ZeroPadding2 (None, 12, 23, 512)  0           stage4_uni
t1_relu2[0][0]
__________________________________________________________________________
_____________________
stage4_unit1_conv2 (Conv2D)     (None, 10, 21, 512)  2359296     zero_paddi
ng2d_29[0][0]
__________________________________________________________________________
_____________________
stage4_unit1_sc (Conv2D)        (None, 10, 21, 512)  131072      stage4_uni
t1_relu1[0][0]
__________________________________________________________________________
_____________________
add_13 (Add)                    (None, 10, 21, 512)  0           stage4_uni
t1_conv2[0][0]
```

```
                                                                    stage4_uni
t1_sc[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit2_bn1 (BatchNormaliz (None, 10, 21, 512)  2048        add_13
[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit2_relu1 (Activation) (None, 10, 21, 512)  0           stage4_uni
t2_bn1[0][0]
__________________________________________________________________________________________
_____________________________
zero_padding2d_30 (ZeroPadding2 (None, 12, 23, 512)  0           stage4_uni
t2_relu1[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit2_conv1 (Conv2D)     (None, 10, 21, 512)  2359296     zero_paddi
ng2d_30[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit2_bn2 (BatchNormaliz (None, 10, 21, 512)  2048        stage4_uni
t2_conv1[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit2_relu2 (Activation) (None, 10, 21, 512)  0           stage4_uni
t2_bn2[0][0]
__________________________________________________________________________________________
_____________________________
zero_padding2d_31 (ZeroPadding2 (None, 12, 23, 512)  0           stage4_uni
t2_relu2[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit2_conv2 (Conv2D)     (None, 10, 21, 512)  2359296     zero_paddi
ng2d_31[0][0]
__________________________________________________________________________________________
_____________________________
add_14 (Add)                    (None, 10, 21, 512)  0           stage4_uni
t2_conv2[0][0]
                                                                    add_13
[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit3_bn1 (BatchNormaliz (None, 10, 21, 512)  2048        add_14
[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit3_relu1 (Activation) (None, 10, 21, 512)  0           stage4_uni
t3_bn1[0][0]
__________________________________________________________________________________________
_____________________________
zero_padding2d_32 (ZeroPadding2 (None, 12, 23, 512)  0           stage4_uni
t3_relu1[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit3_conv1 (Conv2D)     (None, 10, 21, 512)  2359296     zero_paddi
ng2d_32[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit3_bn2 (BatchNormaliz (None, 10, 21, 512)  2048        stage4_uni
t3_conv1[0][0]
__________________________________________________________________________________________
_____________________________
stage4_unit3_relu2 (Activation) (None, 10, 21, 512)  0           stage4_uni
t3_bn2[0][0]
```

```
__________________________________________________________________________________________________
__________________________
zero_padding2d_33 (ZeroPadding2 (None, 12, 23, 512)  0           stage4_uni
t3_relu2[0][0]
__________________________________________________________________________________________________
__________________________
stage4_unit3_conv2 (Conv2D)     (None, 10, 21, 512)  2359296     zero_paddi
ng2d_33[0][0]
__________________________________________________________________________________________________
__________________________
add_15 (Add)                    (None, 10, 21, 512)  0           stage4_uni
t3_conv2[0][0]
                                                                 add_14
[0][0]
__________________________________________________________________________________________________
__________________________
bn1 (BatchNormalization)        (None, 10, 21, 512)  2048        add_15
[0][0]
__________________________________________________________________________________________________
__________________________
relu1 (Activation)              (None, 10, 21, 512)  0           bn1[0][0]
__________________________________________________________________________________________________
__________________________
conv2d (Conv2D)                 (None, 10, 21, 192)  4817088     relu1
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_15 (Conv2D)              (None, 10, 21, 192)  4817088     stage4_uni
t3_bn2[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_30 (Conv2D)              (None, 12, 23, 192)  4817088     zero_paddi
ng2d_30[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization (BatchNorma (None, 10, 21, 192)  768         conv2d
[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_6 (BatchNor (None, 10, 21, 192)  768         conv2d_15
[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_12 (BatchNo (None, 12, 23, 192)  768         conv2d_30
[0][0]
__________________________________________________________________________________________________
__________________________
activation (Activation)         (None, 10, 21, 192)  0           batch_norm
alization[0][0]
__________________________________________________________________________________________________
__________________________
activation_3 (Activation)       (None, 10, 21, 192)  0           batch_norm
alization_6[0][0]
__________________________________________________________________________________________________
__________________________
activation_6 (Activation)       (None, 12, 23, 192)  0           batch_norm
alization_12[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_1 (BatchNor (None, 10, 21, 192)  768         activation
[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_7 (BatchNor (None, 10, 21, 192)  768         activation
```

```
_3[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_13 (BatchNo (None, 12, 23, 192)  768         activation
_6[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_1 (Conv2D)               (None, 10, 21, 192)  37056       batch_norm
alization_1[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_16 (Conv2D)              (None, 10, 21, 192)  37056       batch_norm
alization_7[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_31 (Conv2D)              (None, 12, 23, 192)  37056       batch_norm
alization_13[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_2 (Conv2D)               (None, 10, 21, 192)  331968      conv2d_1
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_3 (Conv2D)               (None, 10, 21, 192)  331968      conv2d_1
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_17 (Conv2D)              (None, 10, 21, 192)  331968      conv2d_16
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_18 (Conv2D)              (None, 10, 21, 192)  331968      conv2d_16
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_32 (Conv2D)              (None, 12, 23, 192)  331968      conv2d_31
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_33 (Conv2D)              (None, 12, 23, 192)  331968      conv2d_31
[0][0]
__________________________________________________________________________________________________
__________________________
multiply (Multiply)             (None, 10, 21, 192)  0           conv2d_2
[0][0]
                                                                 conv2d_3
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_3 (Multiply)           (None, 10, 21, 192)  0           conv2d_17
[0][0]
                                                                 conv2d_18
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_6 (Multiply)           (None, 12, 23, 192)  0           conv2d_32
[0][0]
                                                                 conv2d_33
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_4 (Conv2D)               (None, 10, 21, 192)  37056       multiply
[0][0]
__________________________________________________________________________________________________
```

```
__________________________
conv2d_19 (Conv2D)              (None, 10, 21, 192)  37056       multiply_3
[0][0]
__________________________________________________________________________
__________________________
conv2d_34 (Conv2D)              (None, 12, 23, 192)  37056       multiply_6
[0][0]
__________________________________________________________________________
__________________________
add_16 (Add)                    (None, 10, 21, 192)  0           conv2d_1
[0][0]
                                                                 conv2d_4
[0][0]
__________________________________________________________________________
__________________________
add_19 (Add)                    (None, 10, 21, 192)  0           conv2d_16
[0][0]
                                                                 conv2d_19
[0][0]
__________________________________________________________________________
__________________________
add_22 (Add)                    (None, 12, 23, 192)  0           conv2d_31
[0][0]
                                                                 conv2d_34
[0][0]
__________________________________________________________________________
__________________________
conv2d_5 (Conv2D)               (None, 10, 21, 96)   903264      add_16
[0][0]
__________________________________________________________________________
__________________________
conv2d_20 (Conv2D)              (None, 10, 21, 96)   903264      add_19
[0][0]
__________________________________________________________________________
__________________________
conv2d_35 (Conv2D)              (None, 12, 23, 96)   903264      add_22
[0][0]
__________________________________________________________________________
__________________________
batch_normalization_2 (BatchNor (None, 10, 21, 96)   384         conv2d_5
[0][0]
__________________________________________________________________________
__________________________
batch_normalization_8 (BatchNor (None, 10, 21, 96)   384         conv2d_20
[0][0]
__________________________________________________________________________
__________________________
batch_normalization_14 (BatchNo (None, 12, 23, 96)   384         conv2d_35
[0][0]
__________________________________________________________________________
__________________________
activation_1 (Activation)       (None, 10, 21, 96)   0           batch_norm
alization_2[0][0]
__________________________________________________________________________
__________________________
activation_4 (Activation)       (None, 10, 21, 96)   0           batch_norm
alization_8[0][0]
__________________________________________________________________________
__________________________
activation_7 (Activation)       (None, 12, 23, 96)   0           batch_norm
alization_14[0][0]
__________________________________________________________________________
__________________________
batch_normalization_3 (BatchNor (None, 10, 21, 96)   384         activation
```

```
_1[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_9 (BatchNor (None, 10, 21, 96)   384         activation
_4[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_15 (BatchNo (None, 12, 23, 96)   384         activation
_7[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_6 (Conv2D)               (None, 10, 21, 96)   9312        batch_norm
alization_3[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_21 (Conv2D)              (None, 10, 21, 96)   9312        batch_norm
alization_9[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_36 (Conv2D)              (None, 12, 23, 96)   9312        batch_norm
alization_15[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_7 (Conv2D)               (None, 10, 21, 96)   83040       conv2d_6
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_8 (Conv2D)               (None, 10, 21, 96)   83040       conv2d_6
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_22 (Conv2D)              (None, 10, 21, 96)   83040       conv2d_21
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_23 (Conv2D)              (None, 10, 21, 96)   83040       conv2d_21
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_37 (Conv2D)              (None, 12, 23, 96)   83040       conv2d_36
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_38 (Conv2D)              (None, 12, 23, 96)   83040       conv2d_36
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_1 (Multiply)           (None, 10, 21, 96)   0           conv2d_7
[0][0]
                                                                 conv2d_8
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_4 (Multiply)           (None, 10, 21, 96)   0           conv2d_22
[0][0]
                                                                 conv2d_23
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_7 (Multiply)           (None, 12, 23, 96)   0           conv2d_37
[0][0]
                                                                 conv2d_38
[0][0]
__________________________________________________________________________________________________
```

```
__________________________
conv2d_9 (Conv2D)               (None, 10, 21, 96)   9312        multiply_1
[0][0]
__________________________________________________________________________
__________________________
conv2d_24 (Conv2D)              (None, 10, 21, 96)   9312        multiply_4
[0][0]
__________________________________________________________________________
__________________________
conv2d_39 (Conv2D)              (None, 12, 23, 96)   9312        multiply_7
[0][0]
__________________________________________________________________________
__________________________
add_17 (Add)                    (None, 10, 21, 96)   0           conv2d_6
[0][0]
                                                                 conv2d_9
[0][0]
__________________________________________________________________________
__________________________
add_20 (Add)                    (None, 10, 21, 96)   0           conv2d_21
[0][0]
                                                                 conv2d_24
[0][0]
__________________________________________________________________________
__________________________
add_23 (Add)                    (None, 12, 23, 96)   0           conv2d_36
[0][0]
                                                                 conv2d_39
[0][0]
__________________________________________________________________________
__________________________
conv2d_10 (Conv2D)              (None, 10, 21, 48)   115248      add_17
[0][0]
__________________________________________________________________________
__________________________
conv2d_25 (Conv2D)              (None, 10, 21, 48)   115248      add_20
[0][0]
__________________________________________________________________________
__________________________
conv2d_40 (Conv2D)              (None, 12, 23, 48)   115248      add_23
[0][0]
__________________________________________________________________________
_________________________
batch_normalization_4 (BatchNor (None, 10, 21, 48)   192         conv2d_10
[0][0]
__________________________________________________________________________
_________________________
batch_normalization_10 (BatchNo (None, 10, 21, 48)   192         conv2d_25
[0][0]
__________________________________________________________________________
_________________________
batch_normalization_16 (BatchNo (None, 12, 23, 48)   192         conv2d_40
[0][0]
__________________________________________________________________________
__________________________
activation_2 (Activation)       (None, 10, 21, 48)   0           batch_norm
alization_4[0][0]
__________________________________________________________________________
__________________________
activation_5 (Activation)       (None, 10, 21, 48)   0           batch_norm
alization_10[0][0]
__________________________________________________________________________
__________________________
activation_8 (Activation)       (None, 12, 23, 48)   0           batch_norm
```

```
alization_16[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_5 (BatchNor (None, 10, 21, 48)   192         activation
_2[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_11 (BatchNo (None, 10, 21, 48)   192         activation
_5[0][0]
__________________________________________________________________________________________________
__________________________
batch_normalization_17 (BatchNo (None, 12, 23, 48)   192         activation
_8[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_11 (Conv2D)              (None, 10, 21, 48)   2352        batch_norm
alization_5[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_26 (Conv2D)              (None, 10, 21, 48)   2352        batch_norm
alization_11[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_41 (Conv2D)              (None, 12, 23, 48)   2352        batch_norm
alization_17[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_12 (Conv2D)              (None, 10, 21, 48)   20784       conv2d_11
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_13 (Conv2D)              (None, 10, 21, 48)   20784       conv2d_11
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_27 (Conv2D)              (None, 10, 21, 48)   20784       conv2d_26
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_28 (Conv2D)              (None, 10, 21, 48)   20784       conv2d_26
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_42 (Conv2D)              (None, 12, 23, 48)   20784       conv2d_41
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_43 (Conv2D)              (None, 12, 23, 48)   20784       conv2d_41
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_2 (Multiply)           (None, 10, 21, 48)   0           conv2d_12
[0][0]
                                                                 conv2d_13
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_5 (Multiply)           (None, 10, 21, 48)   0           conv2d_27
[0][0]
                                                                 conv2d_28
[0][0]
__________________________________________________________________________________________________
__________________________
multiply_8 (Multiply)           (None, 12, 23, 48)   0           conv2d_42
```

```
[0][0]
                                                                 conv2d_43
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_14 (Conv2D)              (None, 10, 21, 48)   2352        multiply_2
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_29 (Conv2D)              (None, 10, 21, 48)   2352        multiply_5
[0][0]
__________________________________________________________________________________________________
__________________________
conv2d_44 (Conv2D)              (None, 12, 23, 48)   2352        multiply_8
[0][0]
__________________________________________________________________________________________________
__________________________
add_18 (Add)                    (None, 10, 21, 48)   0           conv2d_11
[0][0]
                                                                 conv2d_14
[0][0]
__________________________________________________________________________________________________
__________________________
add_21 (Add)                    (None, 10, 21, 48)   0           conv2d_26
[0][0]
                                                                 conv2d_29
[0][0]
__________________________________________________________________________________________________
__________________________
add_24 (Add)                    (None, 12, 23, 48)   0           conv2d_41
[0][0]
                                                                 conv2d_44
[0][0]
__________________________________________________________________________________________________
__________________________
global_average_pooling2d_3 (Glo (None, 48)           0           add_18
[0][0]
__________________________________________________________________________________________________
__________________________
global_average_pooling2d_4 (Glo (None, 48)           0           add_21
[0][0]
__________________________________________________________________________________________________
__________________________
global_average_pooling2d_5 (Glo (None, 48)           0           add_24
[0][0]
__________________________________________________________________________________________________
__________________________
tf.concat (TFOpLambda)          (None, 144)          0           global_ave
rage_pooling2d_3[0][0]
                                                                 global_ave
rage_pooling2d_4[0][0]
                                                                 global_ave
rage_pooling2d_5[0][0]
__________________________________________________________________________________________________
__________________________
dropout (Dropout)               (None, 144)          0           tf.concat
[0][0]
__________________________________________________________________________________________________
__________________________
dense (Dense)                   (None, 192)          27840       dropout
[0][0]
__________________________________________________________________________________________________
__________________________
dropout_1 (Dropout)             (None, 192)          0           dense
```

```
[0][0]
_________________________________________________________________
dense_1 (Dense)              (None, 24)                4632      dropout_1
[0][0]
=================================================================
Total params: 41,756,881
Trainable params: 41,737,483
```

In [28]:

```
@tf.function
def _mixup(inp, targ):
    indice = tf.range(len(inp))
    indice = tf.random.shuffle(indice)
    sinp = tf.gather(inp, indice, axis=0)
    starg = tf.gather(targ, indice, axis=0)

    alpha = 0.2
    t = tf.compat.v1.distributions.Beta(alpha, alpha).sample([len(inp)])
    tx = tf.reshape(t, [-1, 1, 1, 1])
    ty = tf.reshape(t, [-1, 1])
    x = inp * tx + sinp * (1-tx)
    y = targ * ty + starg * (1-ty)
#     y = tf.minimum(targ + starg, 1.0) # for multi-label???
    return x, y
```

In [29]:

```
tfrecs = sorted(tf.io.gfile.glob(TRAIN_TFREC + '/*.tfrec'))
parsed_trainval = (tf.data.TFRecordDataset(tfrecs, num_parallel_reads=AUTO
                   .map(_parse_function, num_parallel_calls=AUTOTUNE).unba
                   .filter(_filtTP).enumerate())
```

Stratified 5-Fold

In [30]:

```
indices = []
spid = []
recid = []

for i, sample in tqdm(parsed_trainval.prefetch(AUTOTUNE)):
    indices.append(i.numpy())
    spid.append(sample['species_id'].numpy())
    recid.append(sample['recording_id'].numpy().decode())
```

```
1216it [00:36, 33.18it/s]
```

In [31]:

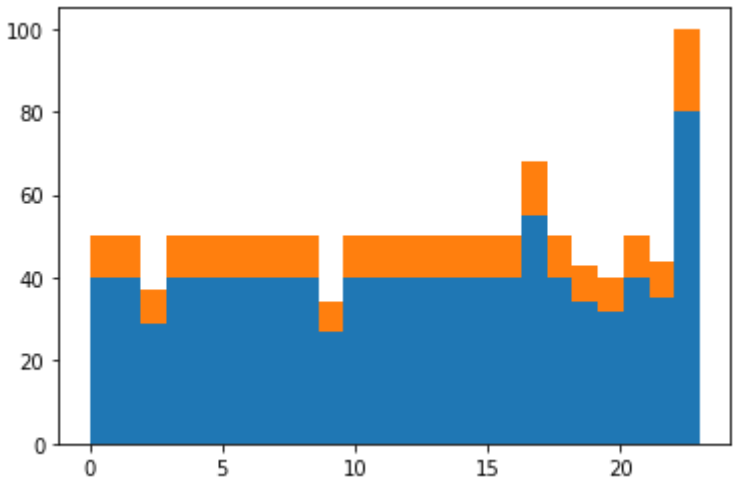
```
table = pd.DataFrame({'indices': indices, 'species_id': spid, 'recording_i
table
```

Out[31]:

	indices	species_id	recording_id
0	0	14	003bec244
1	1	12	2026bced7
2	2	21	422de4e4d
3	3	6	60a493ad4
4	4	13	8080b2283

	indices	species_id	recording_id
...	...	...	...
1211	1211	3	807efd6bb
1212	1212	20	a6610076b
1213	1213	23	c91cae4aa
1214	1214	3	c91cae4aa
1215	1215	1	e755e15ec

In [32]:
```
skf = StratifiedKFold(n_splits=5, random_state=SEED, shuffle=True)
splits = list(skf.split(table.index, table.species_id))

plt.hist([table.loc[splits[0][0], 'species_id'], table.loc[splits[0][1], '
plt.show()
```



In [33]:
```
def create_idx_filter(indice):
    @tf.function
    def _filt(i, x):
        return tf.reduce_any(indice == i)
    return _filt

@tf.function
def _remove_idx(i, x):
    return x
```

Other setup

In [34]:

```
def create_train_dataset(batchsize, train_idx):
    global parsed_trainval
    parsed_train = (parsed_trainval
                    .filter(create_idx_filter(train_idx))
                    .map(_remove_idx))

    dataset = (parsed_train.cache()
        .shuffle(len(train_idx))
        .repeat()
        .map(_cut_wav, num_parallel_calls=AUTOTUNE)
        .map(_wav_to_spec, num_parallel_calls=AUTOTUNE)
        .map(_create_annot, num_parallel_calls=AUTOTUNE)
        .map(_preprocess, num_parallel_calls=AUTOTUNE)
        .batch(batchsize))

    if cfg['model_params']['mixup']:
        dataset = (dataset.map(_mixup, num_parallel_calls=AUTOTUNE)
                    .prefetch(AUTOTUNE))
    else:
        dataset = dataset.prefetch(AUTOTUNE)
    return dataset

def create_val_dataset(batchsize, val_idx):
    global parsed_trainval
    parsed_val = (parsed_trainval
                  .filter(create_idx_filter(val_idx))
                  .map(_remove_idx))

    vdataset = (parsed_val
        .map(_cut_wav_val, num_parallel_calls=AUTOTUNE)
        .map(_wav_to_spec, num_parallel_calls=AUTOTUNE)
        .map(_create_annot, num_parallel_calls=AUTOTUNE)
        .map(_preprocess_val, num_parallel_calls=AUTOTUNE)
        .batch(8*tpu_strategy.num_replicas_in_sync)
        .cache())
    return vdataset
```

Metrics

In [35]:

```
# from https://www.kaggle.com/carlthome/l-lrap-metric-for-tf-keras
@tf.function
def _one_sample_positive_class_precisions(example):
    y_true, y_pred = example

    retrieved_classes = tf.argsort(y_pred, direction='DESCENDING')
    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_true

    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

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)
```

Testset and Inference function

In [36]:

```
def _parse_function_test(example_proto):
    sample = tf.io.parse_single_example(example_proto, feature_description
    wav, _ = tf.audio.decode_wav(sample['audio_wav'], desired_channels=1)

    @tf.function
    def _cut_audio(i):
        _sample = {
            'audio_wav': tf.reshape(wav[i*SR*TIME:(i+1)*SR*TIME], [SR*TIME
            '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
    })

def inference(model):
    tdataset = (tf.data.TFRecordDataset(tf.io.gfile.glob(TEST_TFREC + '/*.
        .map(_parse_function_test, num_parallel_calls=AUTOTUNE).unbatch()
        .map(_wav_to_spec, num_parallel_calls=AUTOTUNE)
        .map(_preprocess_test, num_parallel_calls=AUTOTUNE)
        .batch(128*(60//TIME)).prefetch(AUTOTUNE))

    rec_ids = []
    probs = []
    for inp, rec_id in tqdm(tdataset):
        with tpu_strategy.scope():
            pred = model.predict_on_batch(tf.reshape(inp, [-1, HEIGHT, WID
            prob = tf.sigmoid(pred)
            prob = tf.reduce_max(tf.reshape(prob, [-1, 60//TIME, CLASS_N])

        rec_id_stack = tf.reshape(rec_id, [-1, 60//TIME])
        for rec in rec_id.numpy():
            assert len(np.unique(rec)) == 1
        rec_ids.append(rec_id_stack.numpy()[:,0])
        probs.append(prob.numpy())

    crec_ids = np.concatenate(rec_ids)
    cprobs = np.concatenate(probs)

    sub = pd.DataFrame({
        'recording_id': list(map(lambda x: x.decode(), crec_ids.tolist()))
        **{f's{i}': cprobs[:,i] for i in range(CLASS_N)}
    })
    sub = sub.sort_values('recording_id')

    return sub
```

Now start training!

In [37]:

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

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

    plt.savefig(name)
```

In [38]:

```
def train_and_inference(splits, split_id):
    print(split_id)

    batchsize = cfg['model_params']['batchsize_per_tpu'] * tpu_strategy.nu
    print("batchsize", batchsize)
    loss_fn = cfg['model_params']['loss']['fn'](from_logits=True, **cfg['m

    idx_train_tf = tf.constant(splits[split_id][0])
    idx_val_tf = tf.constant(splits[split_id][1])

    dataset = create_train_dataset(batchsize, idx_train_tf)
    vdataset = create_val_dataset(batchsize, idx_val_tf)

    optimizer = cfg['model_params']['optim']['fn'](**cfg['model_params']['

    with tpu_strategy.scope():
        model = create_model()
        model.compile(optimizer=optimizer, loss=loss_fn, metrics=[LWLRAP(C

    if split_id not in (10,10):#!!!!!!!!!!!! For convenience: If your Col

        history = model.fit(dataset,
                            steps_per_epoch=cfg['model_params']['iteration_p
                            epochs=cfg['model_params']['epoch'],
                            validation_data=vdataset,
                            callbacks=[
                                tf.keras.callbacks.ReduceLROnPlateau(
                                    'val_lwlrap', patience=10
                                ),
                                tf.keras.callbacks.ModelCheckpoint(
                                    filepath=models_path+'model_best_%d.h5' %
                                    save_weights_only=True,
                                    monitor='val_lwlrap',
                                    mode='max',
                                    save_best_only=True),
                            ]
                            )
        plot_history(history, 'history_%d.png' % split_id)
        best_score = max(history.history['val_lwlrap'])
        print (best_score)
    ### inference ###

    model.load_weights(models_path+'model_best_%d.h5' % split_id)
    sub=inference(model)
    del model
    gc.collect()
    return sub,best_score
```

In [39]:
```
# train and inference
# sub, _ = train_and_inference(splits, 0)

# N-fold ensemble


""" Delete this line to start training the model
print(SEED)
train_n=0
df = pd.DataFrame(columns=["train_n",'split_id','best_score','CSV','SEED'])
for split_id in range(len(splits)):
    sub, best_score=train_and_inference(splits, split_id)
    sub.set_index('recording_id').to_csv(models_path+f"submission_train_n_{train_n}_split_id_{split_id}.csv")
    df = df.append({'train_n': train_n,'split_id': split_id,'best_score': best_score,'CSV': f"submission_train_n_{train_n}_split_id_{split_id}.csv",'SEED':SEED}, ignore_index=True)
df.to_csv(models_path+f"train_n_{train_n}.csv", index=False)
#"""
```

Out[39]:
```
' Delete this line to start training the model\nprint(SEED)\ntrain_n=0\ndf
= pd.DataFrame(columns=["train_n",\'split_id\',\'best_score\',\'CSV\',\'SEE
D\'])\nfor split_id in range(len(splits)):\n    sub, best_score=train_and_i
nference(splits, split_id)\n    sub.set_index(\'recording_id\').to_csv(mode
ls_path+f"submission_train_n_{train_n}_split_id_{split_id}.csv", index=Fals
e)\n    df = df.append({\'train_n\': train_n,\'split_id\': split_id,\'best_
score\': best_score,\'CSV\': f"submission_train_n_{train_n}_split_id_{split
_id}.csv",\'SEED\':SEED}, ignore_index=True)\ndf.to_csv(models_path+f"train
_n_{train_n}.csv", index=False)\n#'
```

In [40]:
```
#sub.describe()
```

If you like my notebook don't forget to upvoted it

If you have questions then ask, I will help as I can

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.
```

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 i
    # 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
```

```
        op_to_select = tf.random.uniform([], maxval=len(available_ops), dt
        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

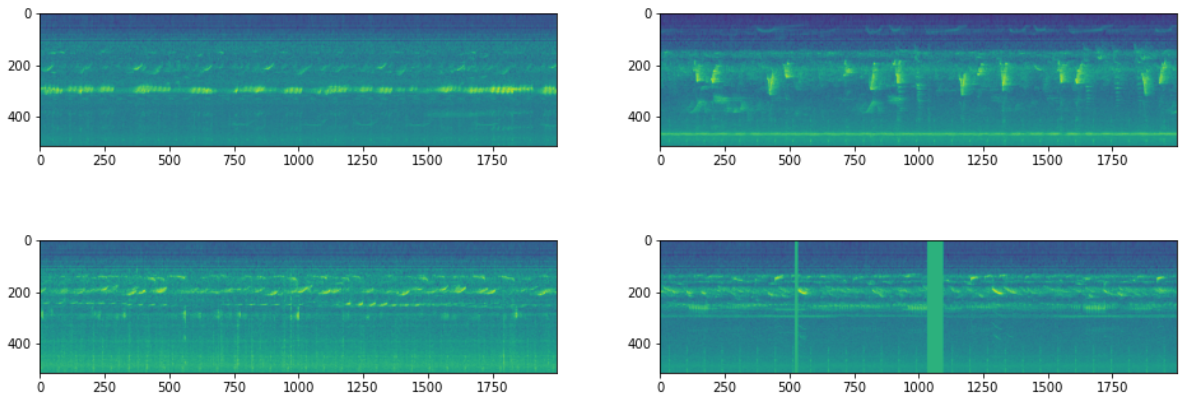
In [10]:

```
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
    })
```

In [11]:
```
def load_dataset(filenames, labeled = True, ordered = False , training = T
    # 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

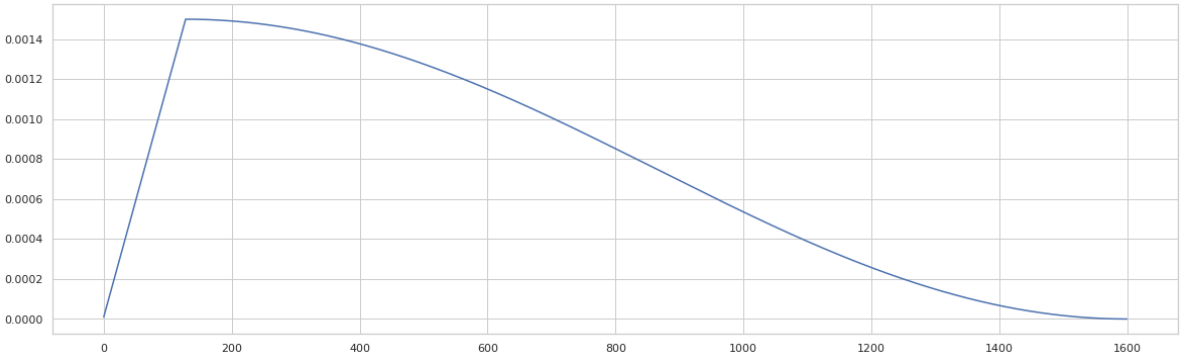
In [14]:

```
# 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_true

    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)
```

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 t
                             '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>]
```



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_rat
                                      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

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.u
    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
_________________________________________________________________
```

```
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
```

```
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
```

```
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
```

 [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
```



```
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
```

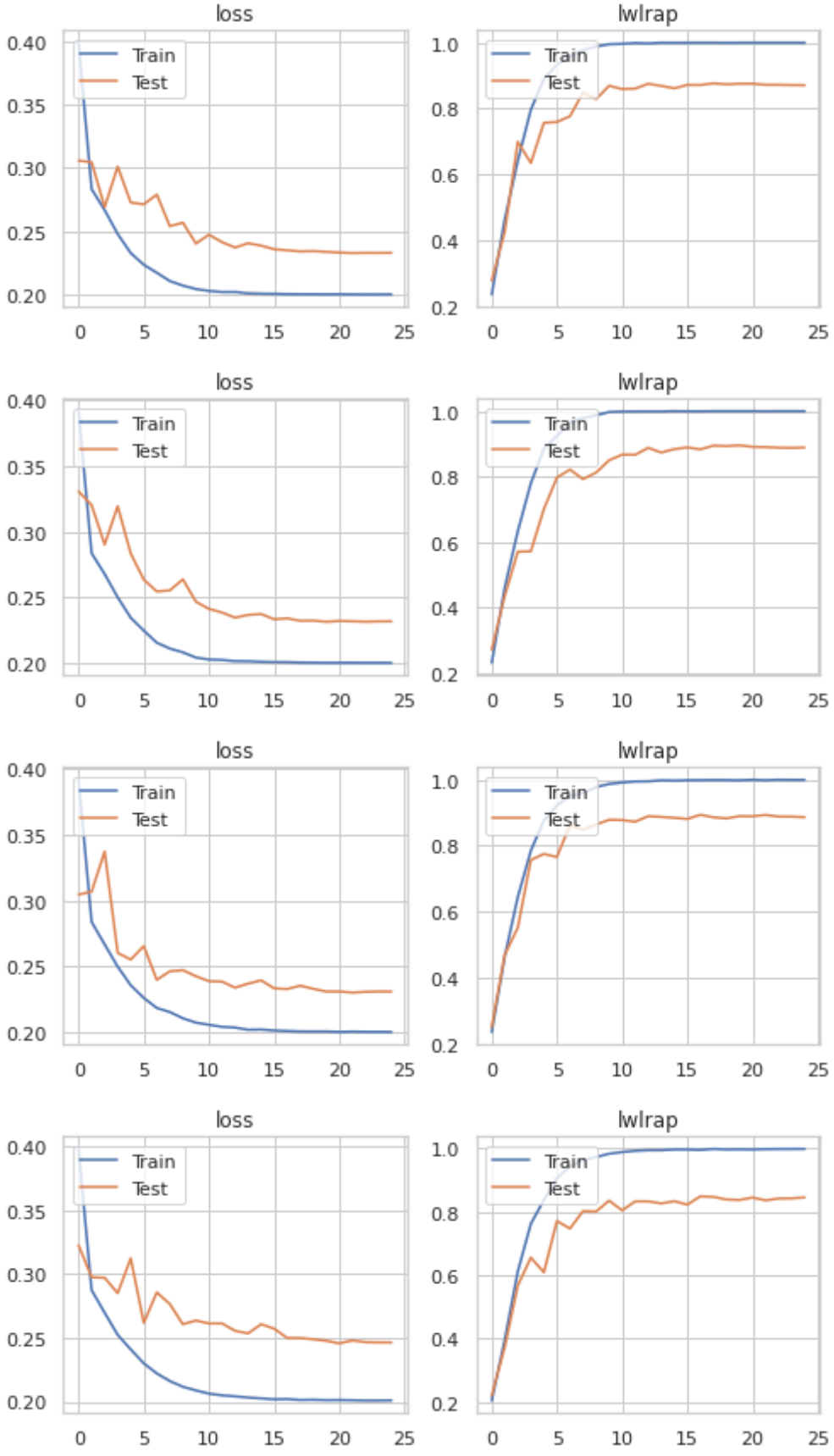
```
s: 0.2465
Epoch 25/25
Restoring model weights from the end of the best epoch.
64/64 - 39s - lwlrap: 0.9987 - loss: 0.2010 - val_lwlrap: 0.8455 - val_los
s: 0.2464
Epoch 00025: early stopping
#### FOLD 4 OOF Accuracy = 0.849

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)
```



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
```

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 [ ]:

Comparative Method - Part(A)

Rainforest Connection Species Audio Detection

By: Somayyeh Gholami & Mehran Kazeminia

Description:

- At the end of the challenge, Mr. @meaninglesslives shared his notebook. He won third place in the challenge. The score of the notebook published in the first version is "public score 0.96171 and private score 0.96460". Thanks for sharing the results, we congratulate him too.

https://www.kaggle.com/meaninglesslives/rfcx-minimal?scriptVersionId=54514070

- Then Mr. @cdeotte released another notebook and with a great trick, raised the previous notebook's private score above 0.970. We also thank him for sharing this trick.

https://www.kaggle.com/cdeotte/rainforest-post-process-lb-0-970

https://www.kaggle.com/c/rfcx-species-audio-detection/discussion/220389

An important question:

Does this trick improve all the results (all the columns)?

No, of course the results of some columns are getting worse.

This means that the results of some columns get very good and the results of some columns get worse, but in this challenge (and usually) the overall results improve.

To prove this, we wrote this notebook and share it with you. Our method is very simple. We first identified nine columns, the results of which will be reduced by performing this trick.

S0, S1, S4, S8, S11, S14, S18, S20, S21

Then we transferred the results of these columns from the original notebook (the first notebook) and replaced these results exactly with the results of the second notebook, and finally saved

the entire result in the "d" file. That is, in file "d", a trick is applied for the results of 15 columns and no trick is applied for the results of 9 columns. The scores of the "d" file are as follows:

"d" : [(Private Score: 0.97915) , (Public Score: 0.97373)]

As you can see, this trick is not good for the results of these nine columns, and we got a much better score with the results of the original notebook (first version). Please note that in order to be able to compare, we used exactly the results of the first version of the original notebook.

In the end, we were able to easily improve the results once again with our own method. We saved the final results in the "e" file. The scores of the "e" file are as follows:

"e" : [(Private Score: 0.98022) , (Public Score: 0.97490)]

If you find this work useful, please don't forget upvoting :)

Import & Data Set

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline

# _____________________________

sub961 = pd.read_csv("../input/rain961/RAIN961.csv")

sub968 = pd.read_csv("../input/rainforest-post-process-lb-0-970/submission_
```

Functions

In [2]:
```
def generate(main, support, coeff):
    g1 = main.copy()
    g2 = main.copy()
    g3 = main.copy()
    g4 = main.copy()

    for i in main.columns[1:]:
        lm, Is = [], []
        lm = main[i].tolist()
        ls = support[i].tolist()

        res1, res2, res3, res4 = [], [], [], []
        for j in range(len(main)):
            res1.append(max(lm[j] , ls[j]))
            res2.append(min(lm[j] , ls[j]))
            res3.append((lm[j] + ls[j]) / 2)
            res4.append((lm[j] * coeff) + (ls[j] * (1.- coeff)))

        g1[i] = res1
        g2[i] = res2
        g3[i] = res3
        g4[i] = res4

    return g1,g2,g3,g4
```

In [3]:
```
def generate1(main, support, coeff):

    g = main.copy()
    for i in main.columns[1:]:

        res = []
        lm, Is = [], []
        lm = main[i].tolist()
        ls = support[i].tolist()

        for j in range(len(main)):
            res.append((lm[j] * coeff[i]) + (ls[j] * (1.- coeff[i])))
        g[i] = res

    return g
```

In [4]:
```
def drawing(main, support, generated, column_number):

    X  = main.iloc[:, column_number]
    Y1 = support.iloc[:, column_number]
    Y2 = generated.iloc[:, column_number]

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize=(8, 8), facecolor='lightgray')
    plt.title(f'\nOn the X axis >>> main\n\nOn the Y axis >>> support\n')
    plt.scatter(X, Y1, s=3)
    plt.show()

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize=(8, 8), facecolor='lightgray')
    plt.title(f'\nOn the X axis >>> main\n\nOn the Y axis >>> generated\n'
    plt.scatter(X, Y2, s=3)
    plt.show()
```

In [5]:
```
def drawing1(main, support, generated, column_number):

    X  = main.iloc[:, column_number]
    Y1 = support.iloc[:, column_number]
    Y2 = generated.iloc[:, column_number]

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize=(8, 8), facecolor='lightgray')
    plt.title(f'\nBlue | X axis >> main | Y axis >> support\n\nOrange | X a

    plt.scatter(X, Y1, s=3)
    plt.scatter(X, Y2, s=3)

    plt.show()
```

Comparative Method

In [6]:
```
# print(sub968.mean() , sub961.mean())

m1 = sub968.mean() + sub961.mean()

m1mean = m1.mean()

m2 = m1 / m1mean

m2
```

Out[6]:
```
s0     0.809025
s1     1.422532
s2     0.534986
s3     5.994374
s4     0.348289
s5     0.550068
s6     0.085884
s7     1.907275
s8     0.393510
s9     0.343576
s10    0.262067
s11    1.039271
s12    2.413379
s13    0.263196
s14    0.757137
s15    2.120569
s16    0.454357
s17    0.213483
s18    2.840838
s19    0.085247
s20    0.167267
s21    0.063781
s22    0.200014
s23    0.729876
dtype: float64
```

In [7]:
```
m3 = m2.copy()
for k in range(24):
    m3[k] = 1.00

# m3
```

In [8]:
```
m4 = m3.copy()

m4[0]   = 0.00

m4[1]   = 0.00

m4[4]   = 0.00

m4[8]   = 0.00

m4[11]  = 0.00

m4[14]  = 0.00

m4[18]  = 0.00

m4[20]  = 0.00

m4[21]  = 0.00
```

Result

[(Private Score: 0.97892) , (Public Score: 0.97309)]

In [9]:
```
m4[15]   = 1.30

m4
```

Out[9]:
```
s0     0.0
s1     0.0
s2     1.0
s3     1.0
s4     0.0
s5     1.0
s6     1.0
s7     1.0
s8     0.0
s9     1.0
s10    1.0
s11    0.0
s12    1.0
s13    1.0
s14    0.0
s15    1.3
s16    1.0
s17    1.0
s18    0.0
s19    1.0
s20    0.0
s21    0.0
```

```
s23    1.0
dtype: float64
```

Result

[(Private Score: 0.97915) , (Public Score: 0.97373)]

In [10]:
```
d = generate1(sub968, sub961, m4)
```

In [11]:
```
sub968.describe()
```

Out[11]:

	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1.992000
mean	0.110630	0.198951	0.058836	0.975920	0.027766	0.049287	3.443965
std	0.298508	0.369604	0.221785	0.095970	0.135427	0.195465	5.238256
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000017	0.000222	0.000010	0.997896	0.000018	0.000032	5.145849
50%	0.000137	0.002012	0.000070	0.999792	0.000089	0.000219	4.882412
75%	0.002537	0.082179	0.000732	0.999958	0.000677	0.001828	2.530490
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 24 columns

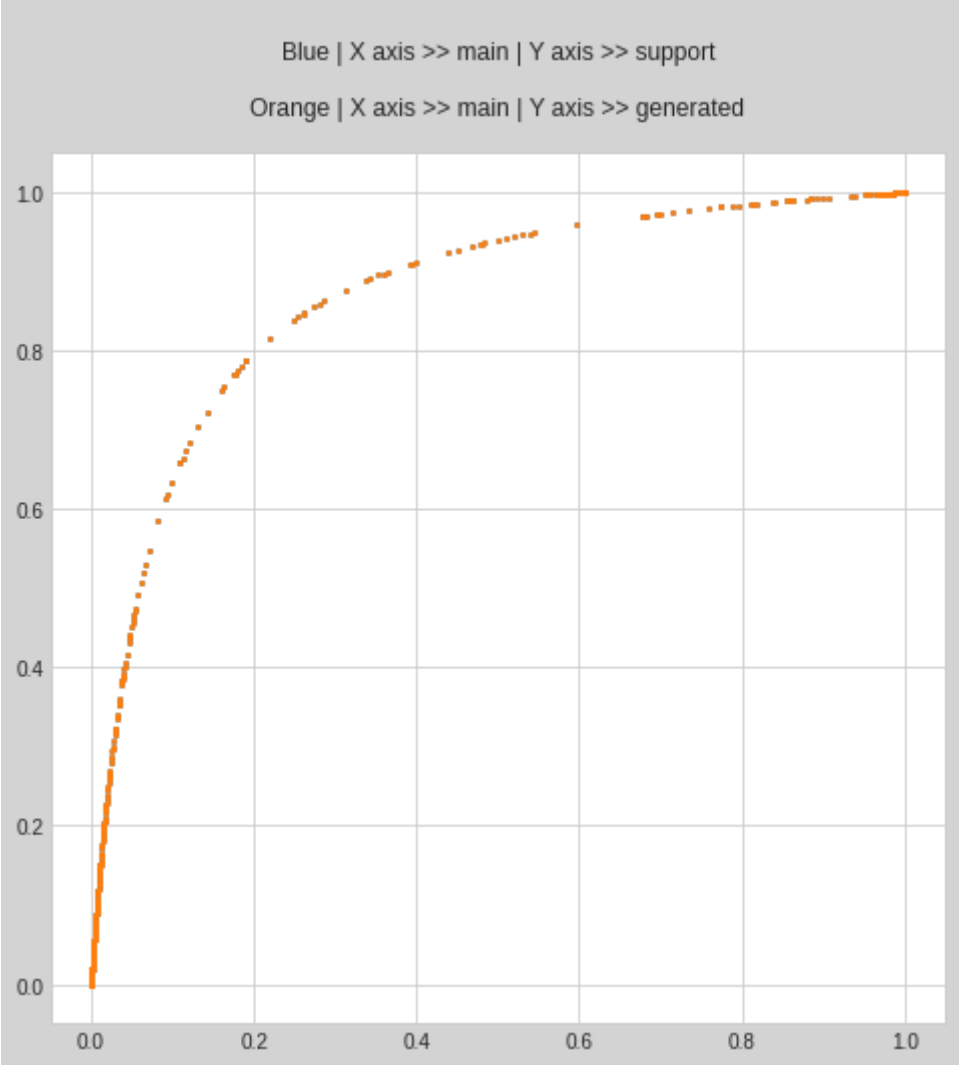
In [12]:
```
sub961.describe()
```

Out[12]:

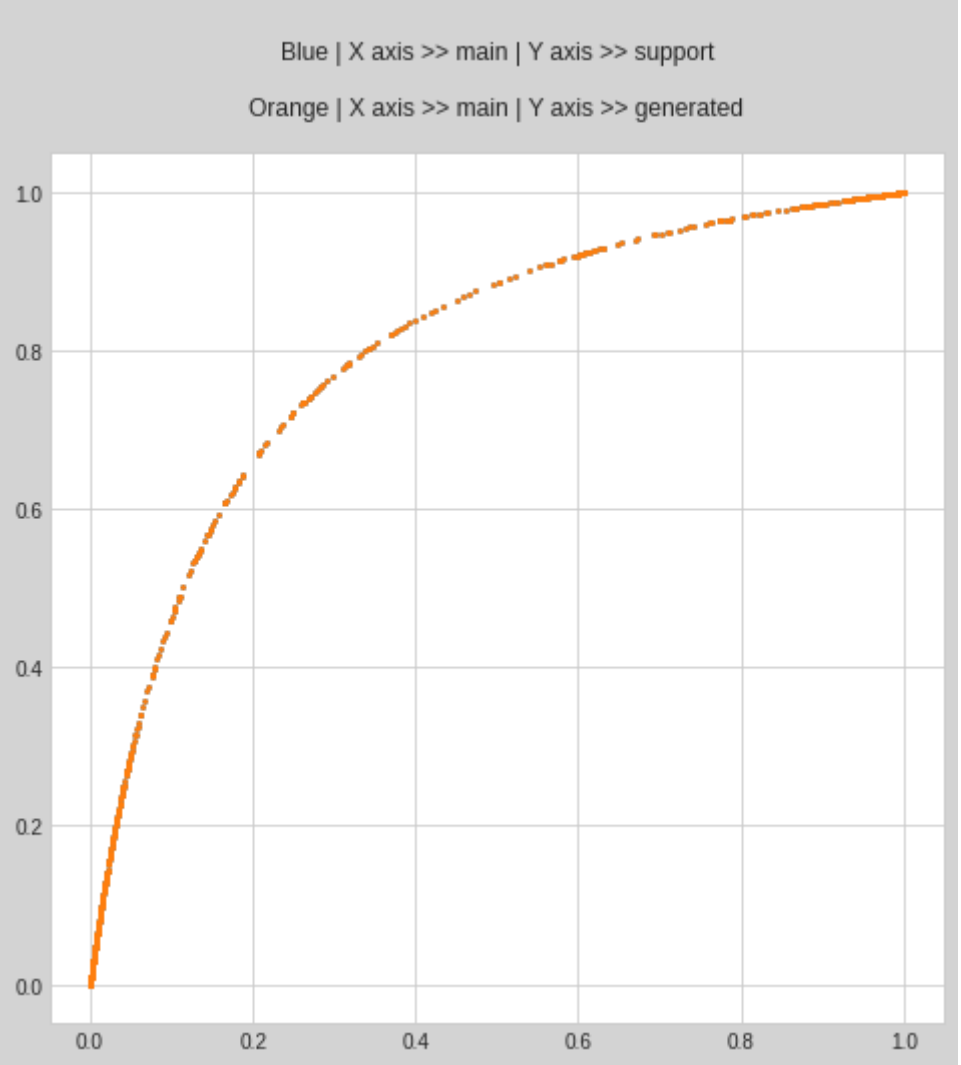
	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1.992000e+03	1992.000000	1992.000000	1992.000000	1992.00
mean	0.148069	0.255927	1.122349e-01	0.940881	0.083605	0.126606	0.02
std	0.324055	0.394491	2.696748e-01	0.166613	0.221616	0.264003	0.09
min	0.000001	0.000031	6.896084e-07	0.038022	0.000003	0.000004	0.00
25%	0.000264	0.001760	4.389717e-04	0.987995	0.000638	0.001514	0.00
50%	0.002151	0.015521	3.022099e-03	0.998800	0.003190	0.010141	0.00
75%	0.038402	0.411339	3.083062e-02	0.999755	0.023652	0.078918	0.01
max	0.999999	0.999999	1.000000e+00	0.999999	0.999976	0.999997	0.99

8 rows × 24 columns

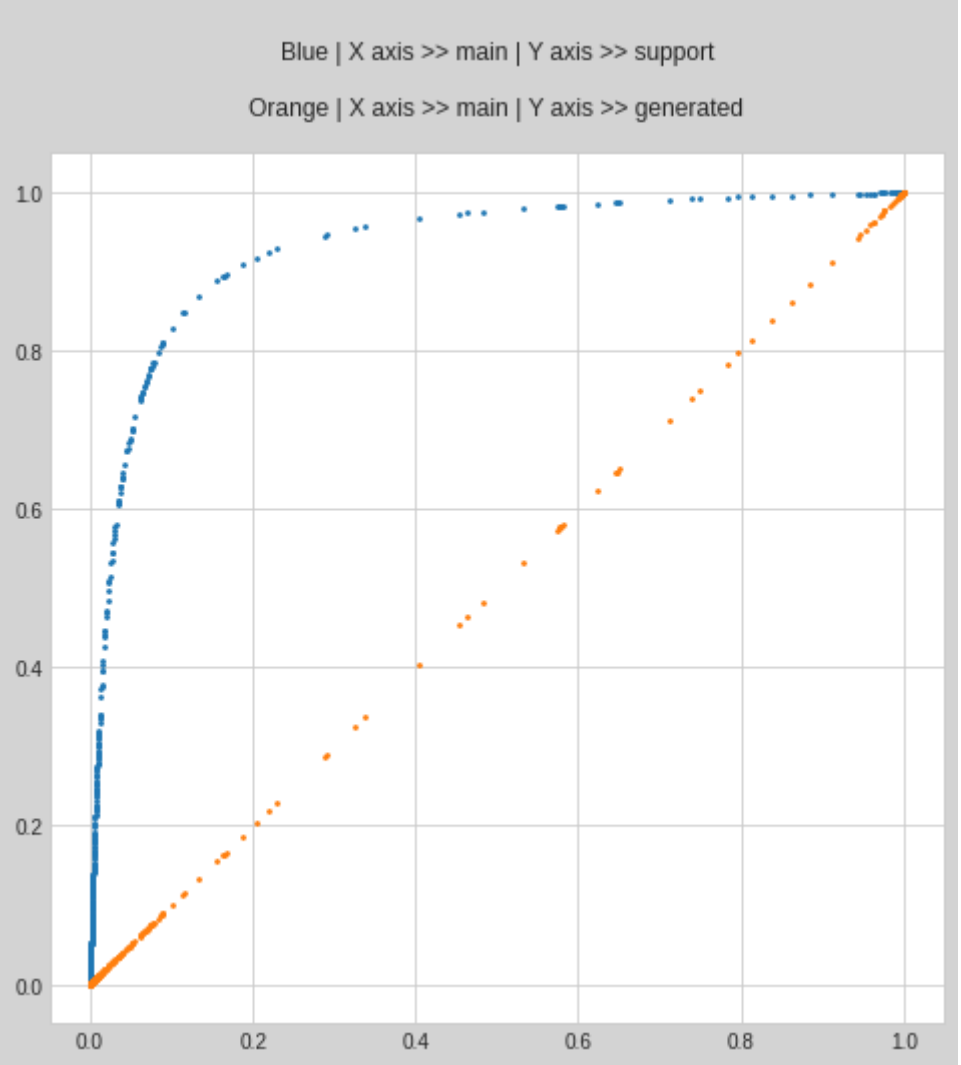
In [13]:
```
drawing1(sub968, sub961, d, 1)
```



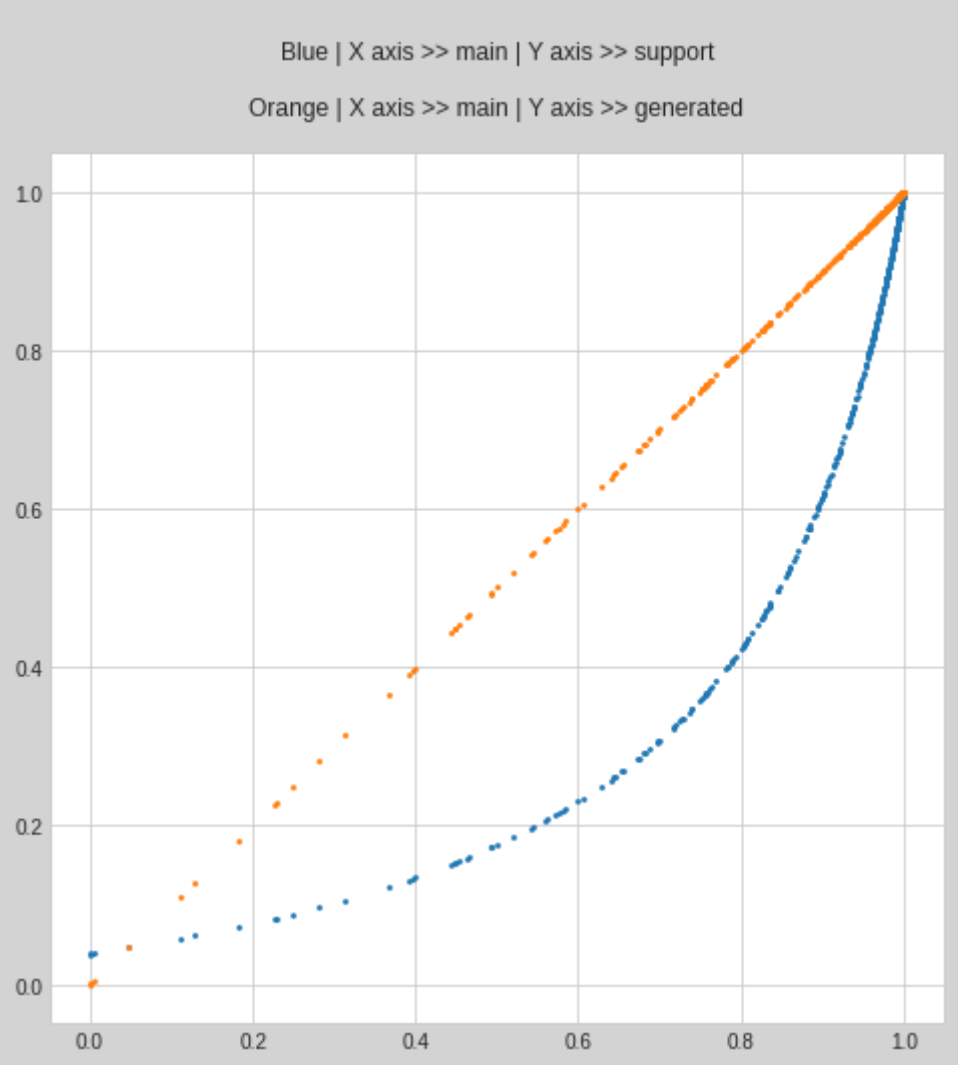
In [14]:
```
drawing1(sub968, sub961, d, 2)
```



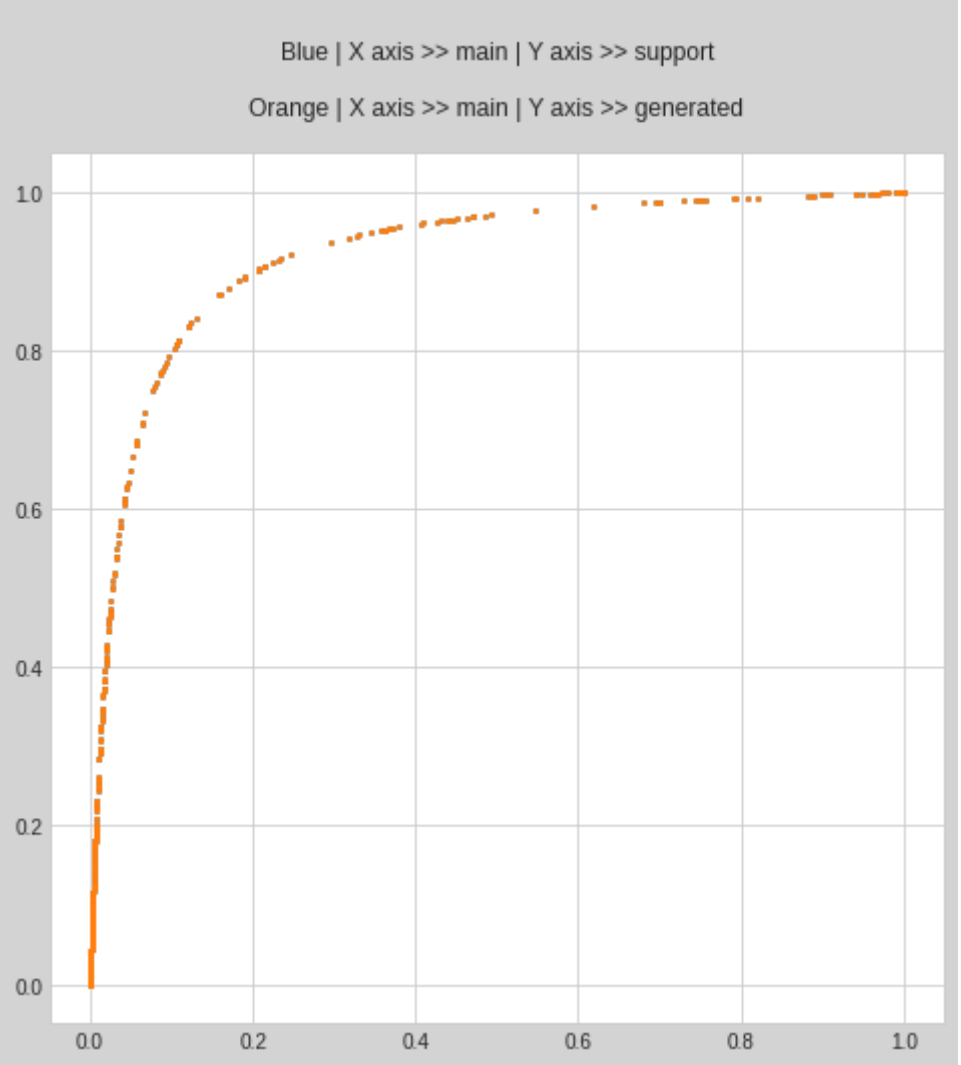
In [15]:
```
drawing1(sub968, sub961, d, 3)
```



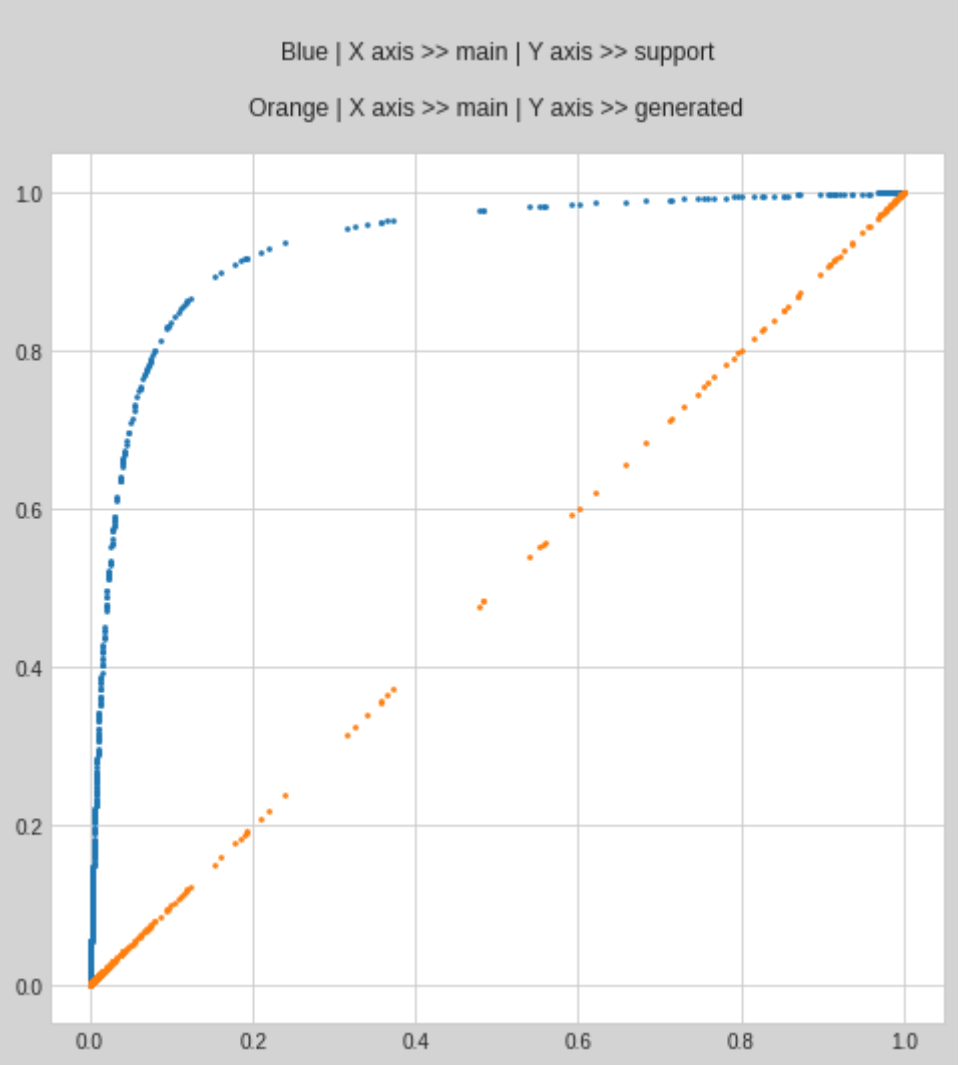
In [16]:
```
drawing1(sub968, sub961, d, 4)
```



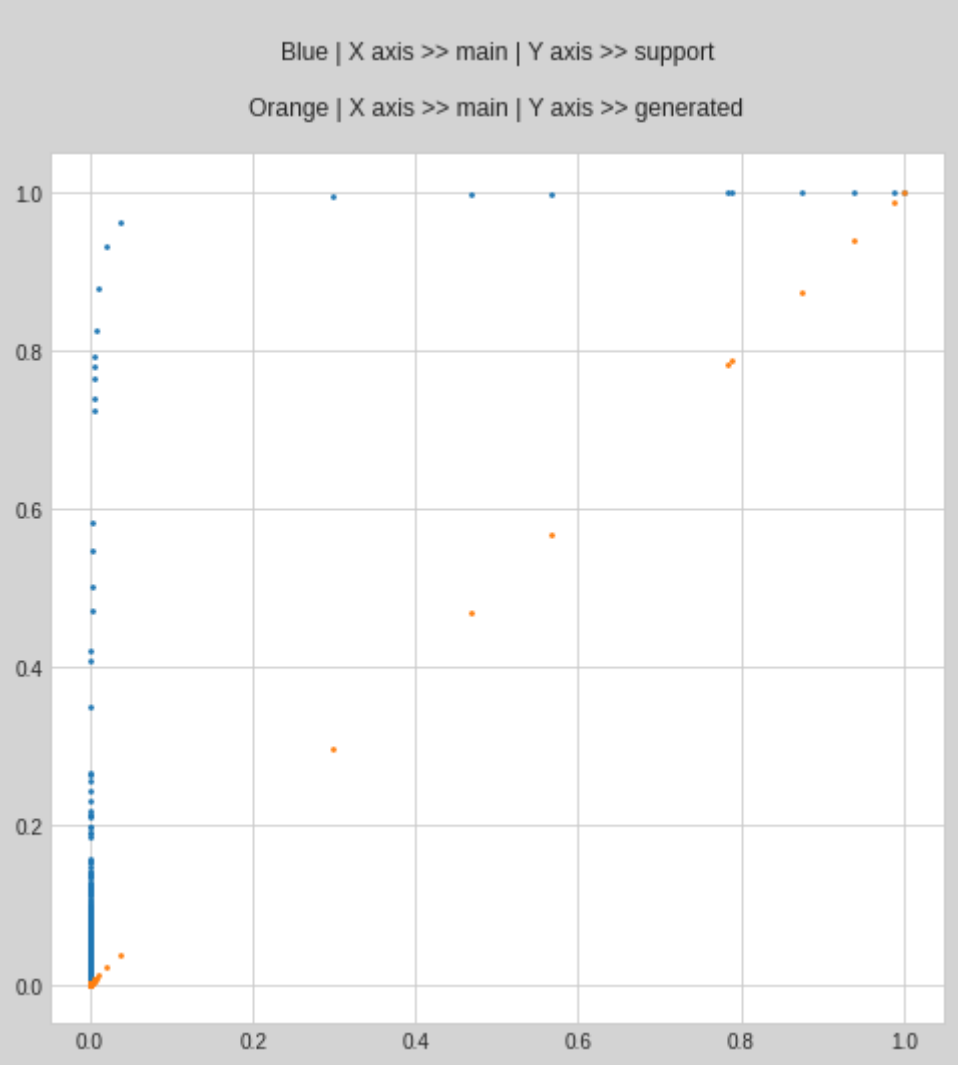
In [17]:
```
drawing1(sub968, sub961, d, 5)
```



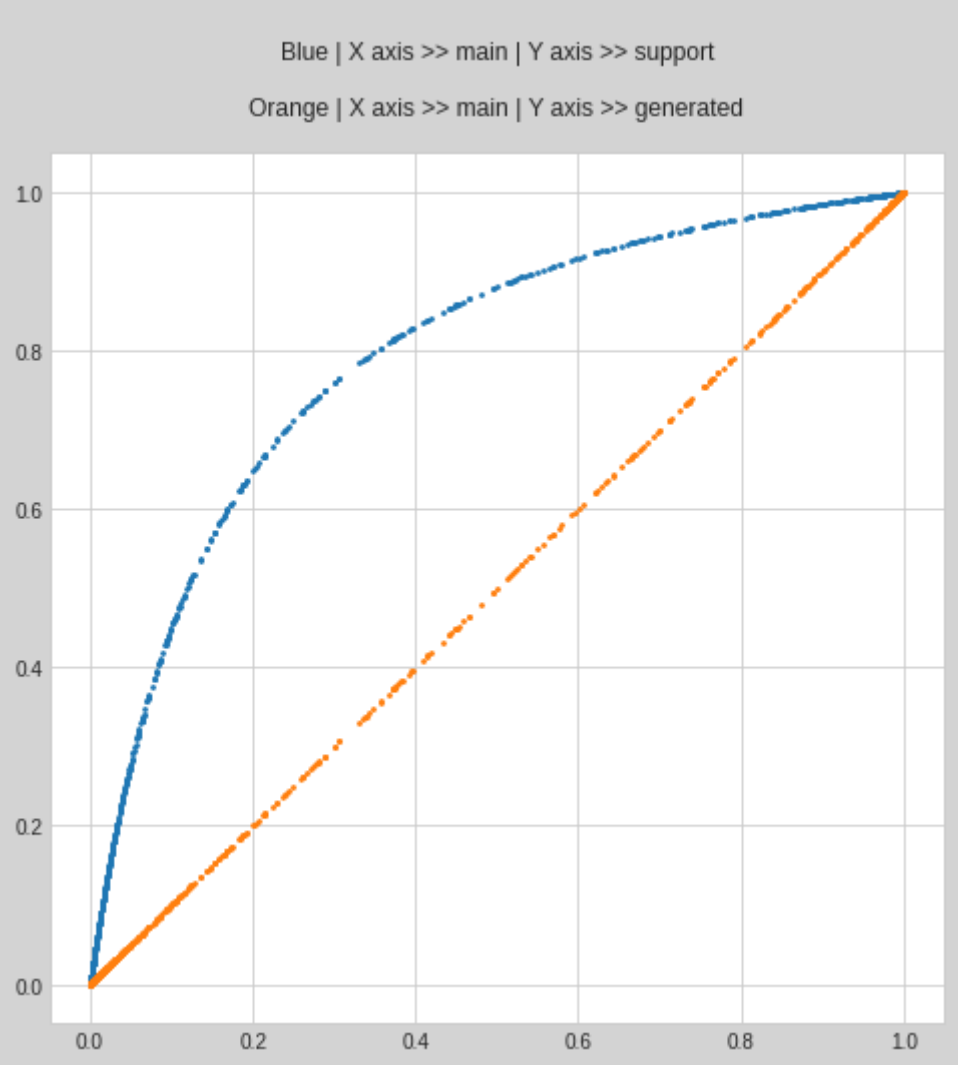
In [18]:
```
drawing1(sub968, sub961, d, 6)
```



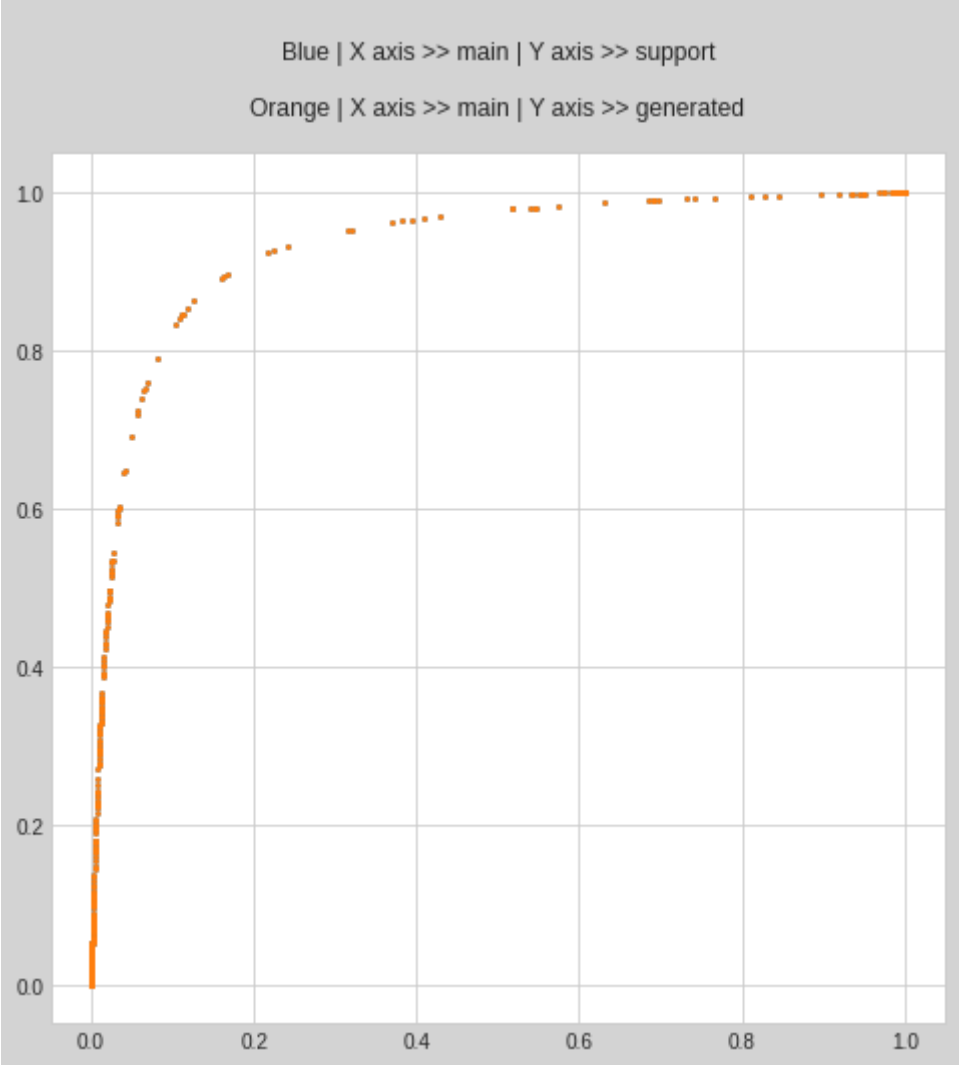
In [19]:
```
drawing1(sub968, sub961, d, 7)
```



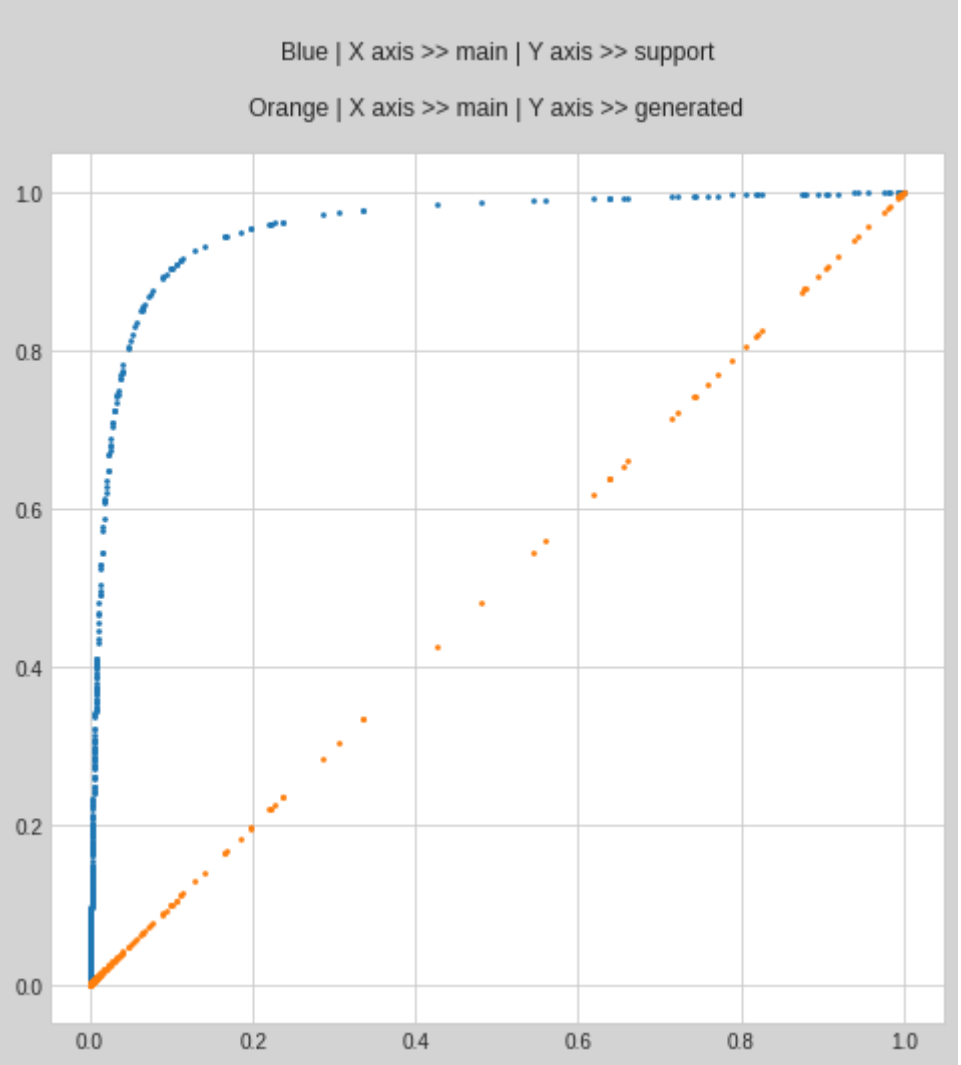
In [20]:
```
drawing1(sub968, sub961, d, 8)
```



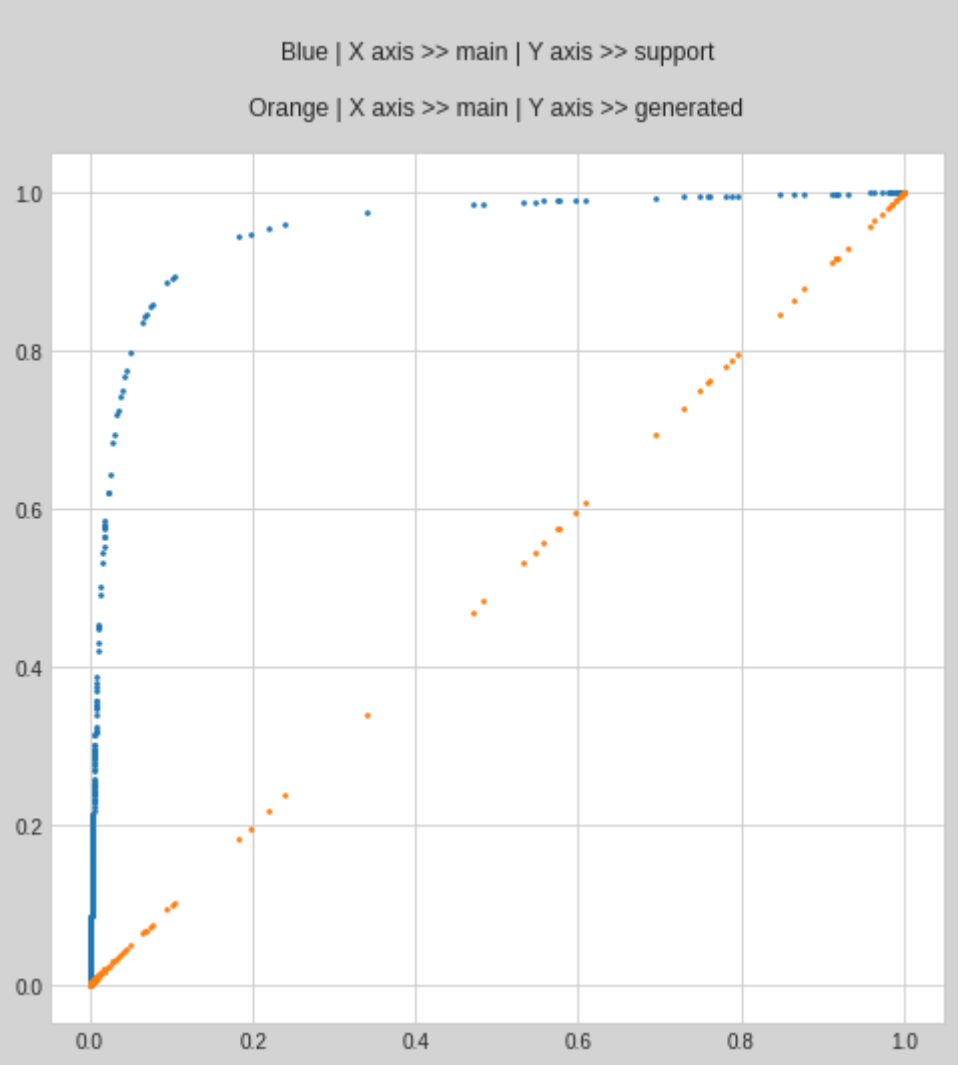
In [21]:
```
drawing1(sub968, sub961, d, 9)
```



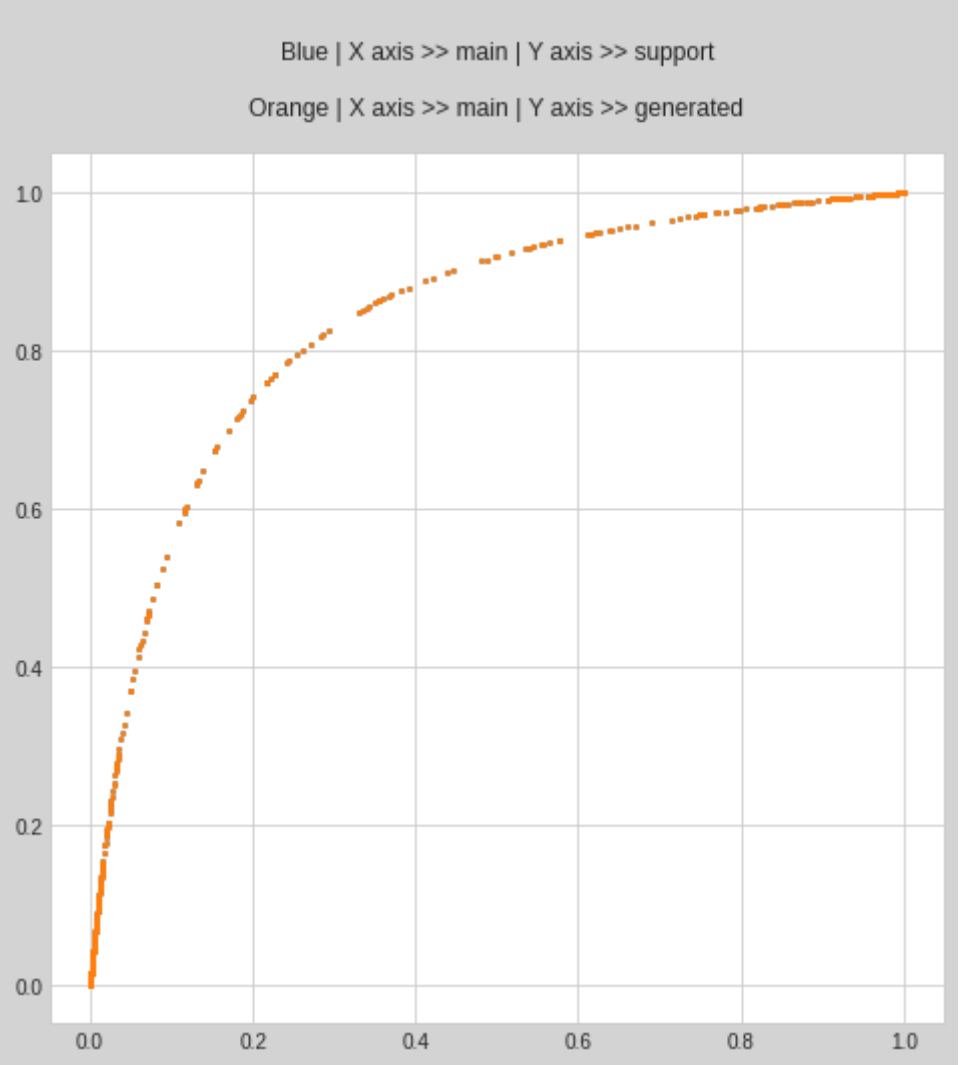
In [22]:
```
drawing1(sub968, sub961, d, 10)
```



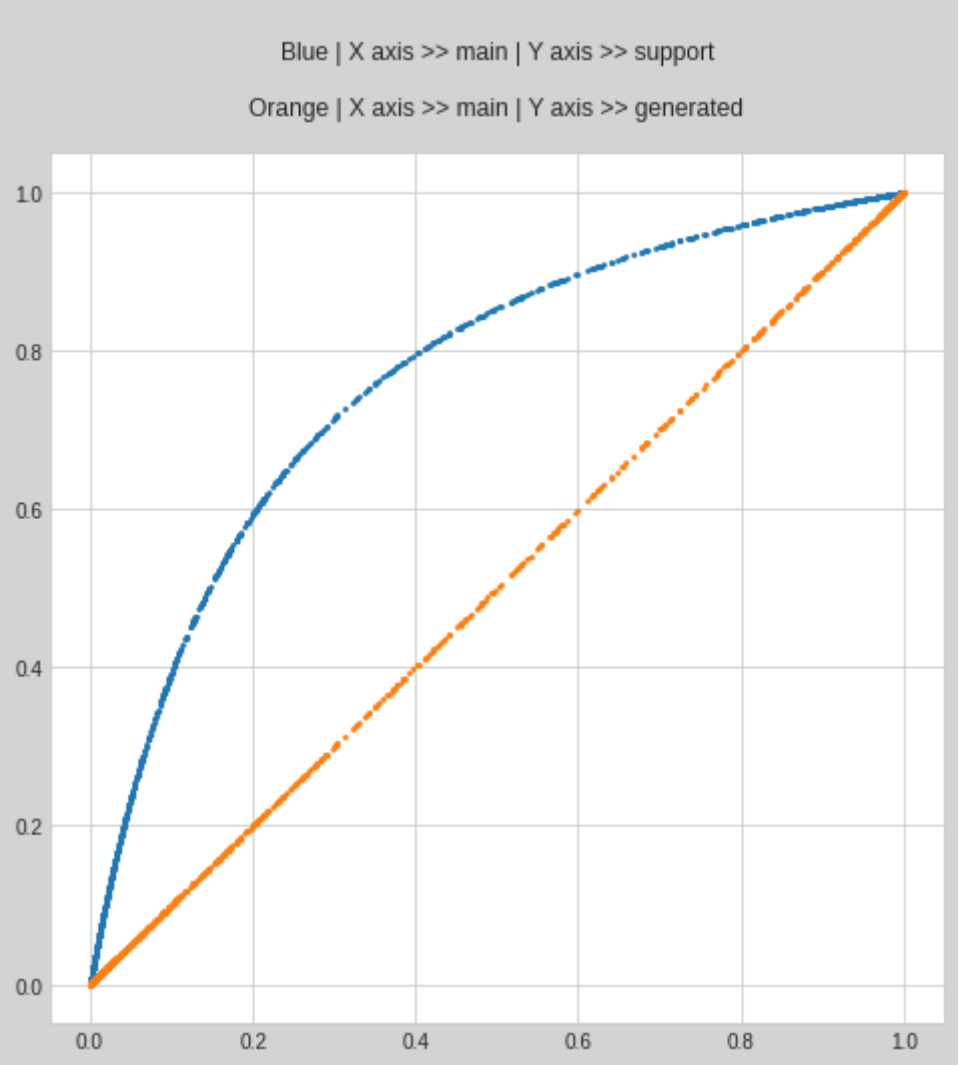
In [23]:
```
drawing1(sub968, sub961, d, 11)
```



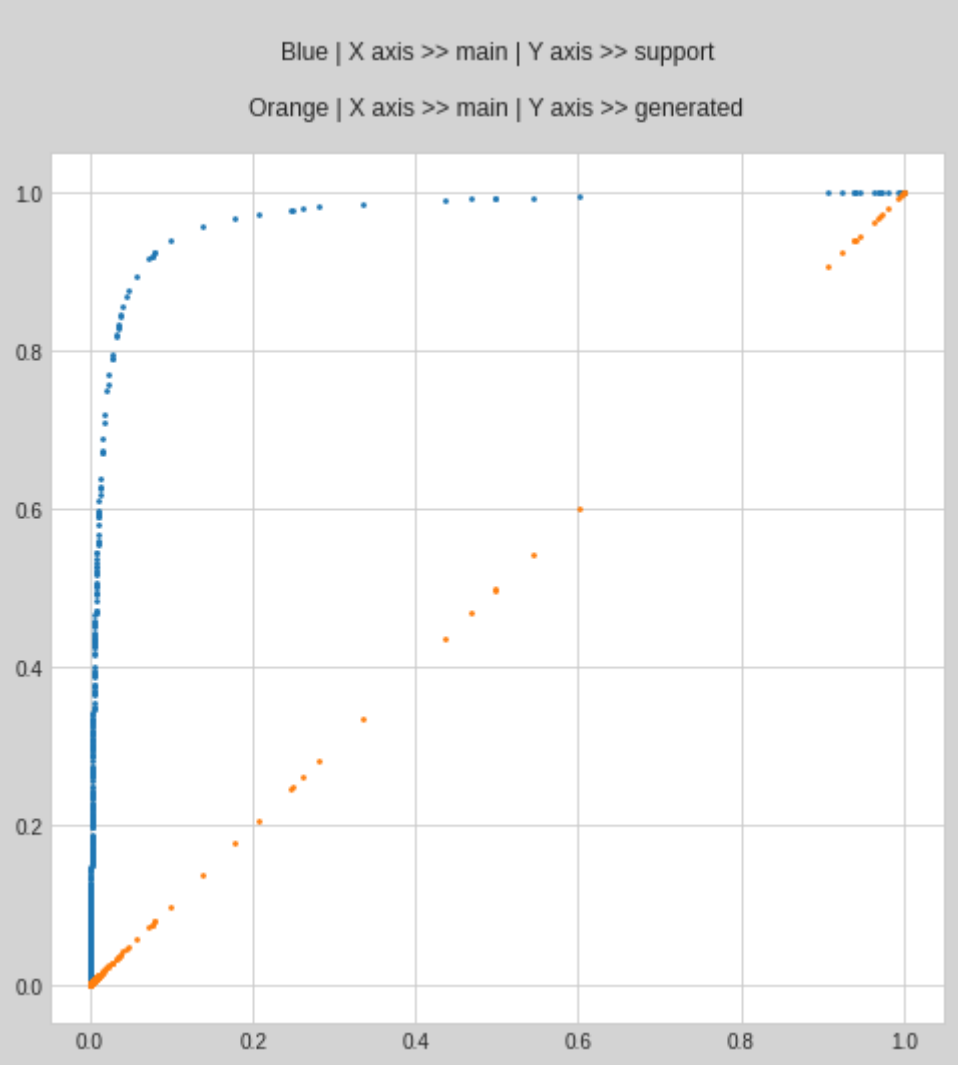
In [24]:
```
drawing1(sub968, sub961, d, 12)
```



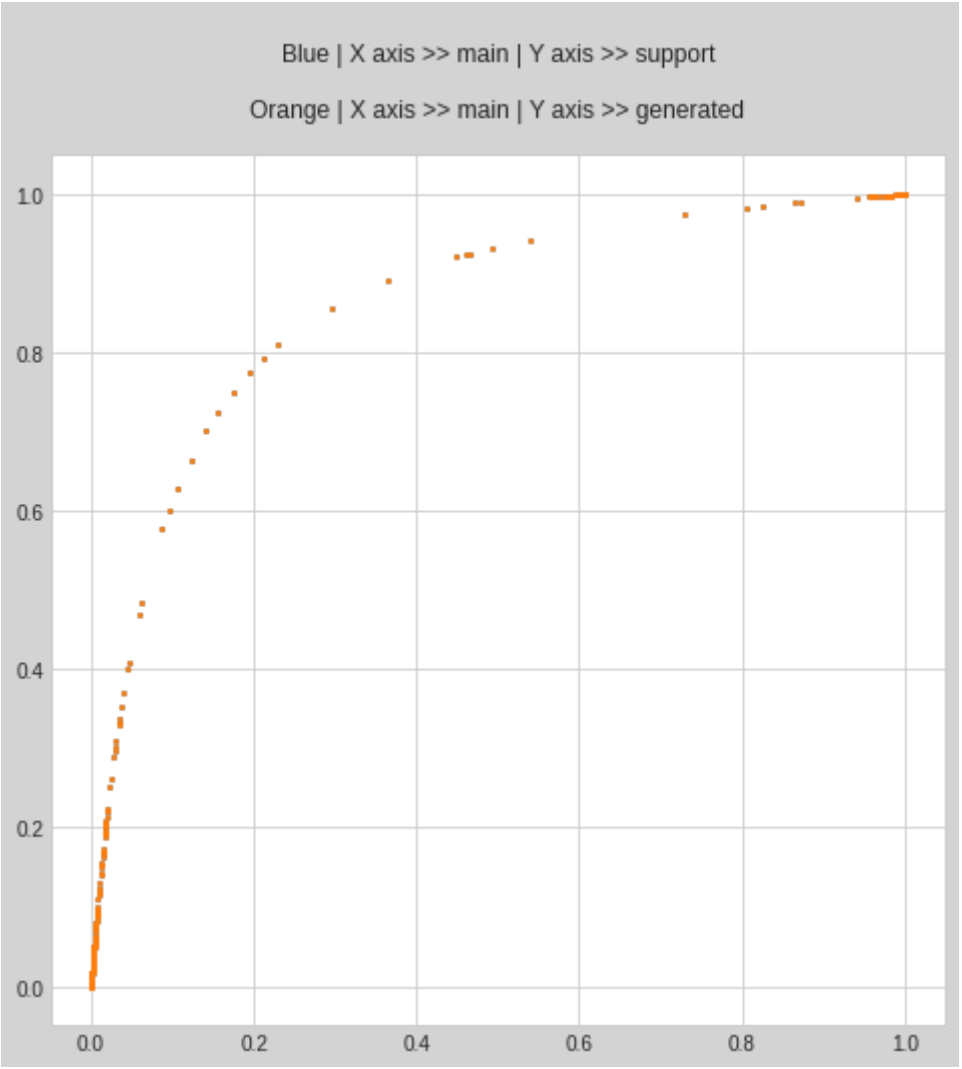
In [25]:
```
drawing1(sub968, sub961, d, 13)
```



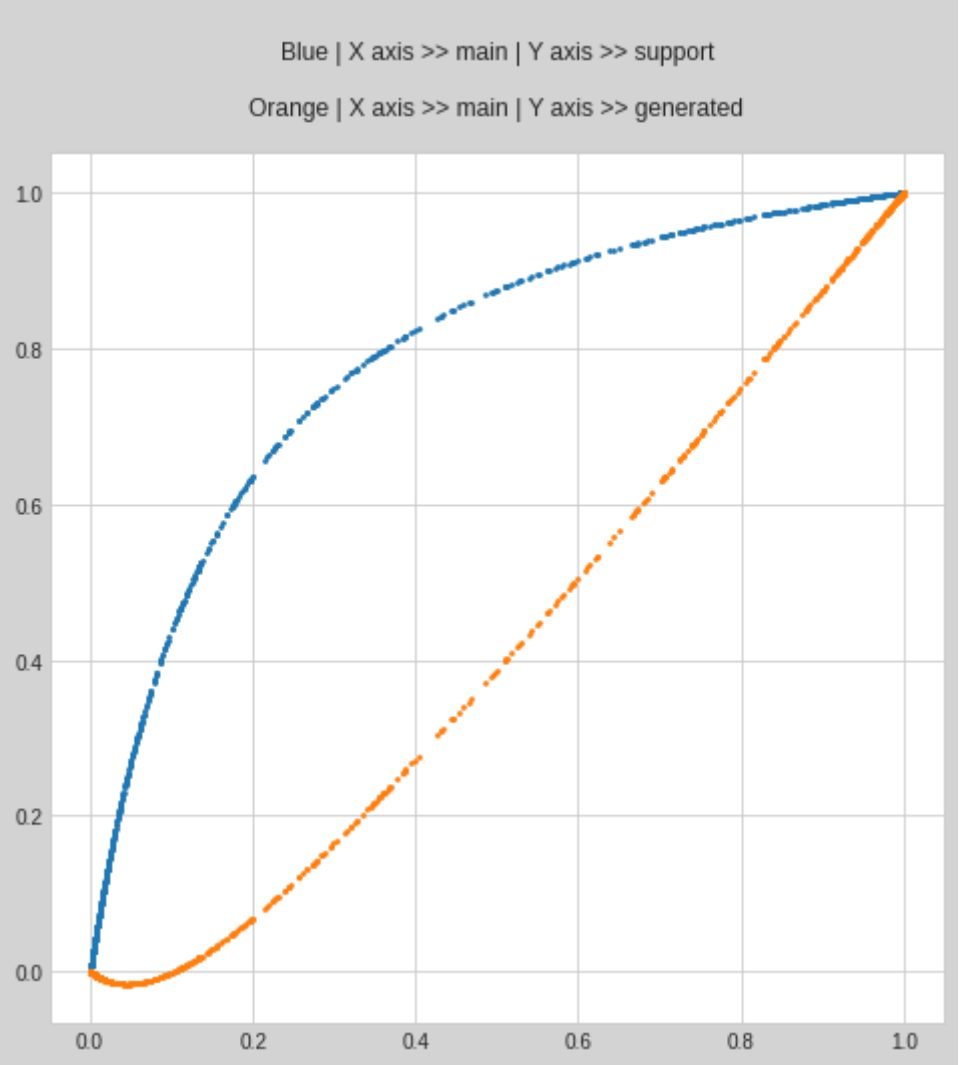
In [26]:
```
drawing1(sub968, sub961, d, 14)
```



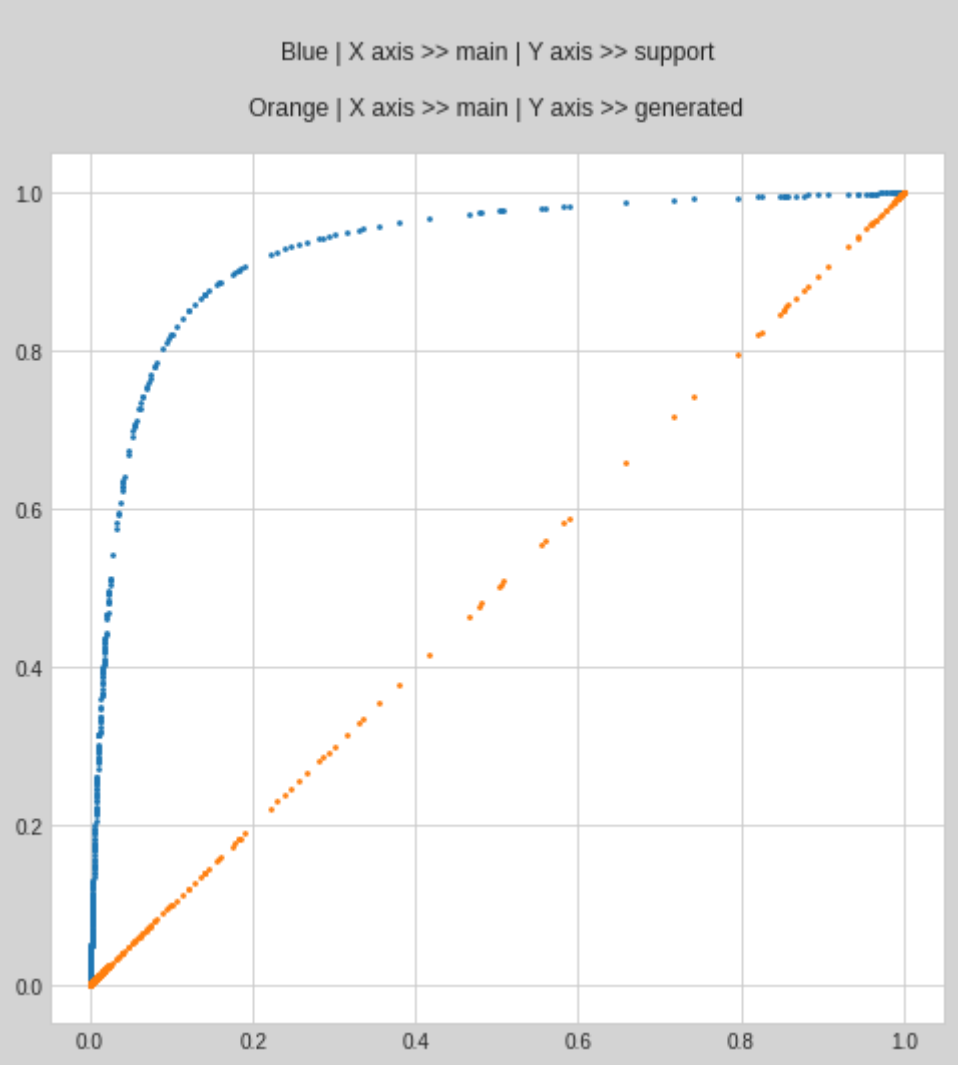
In [27]:
```
drawing1(sub968, sub961, d, 15)
```



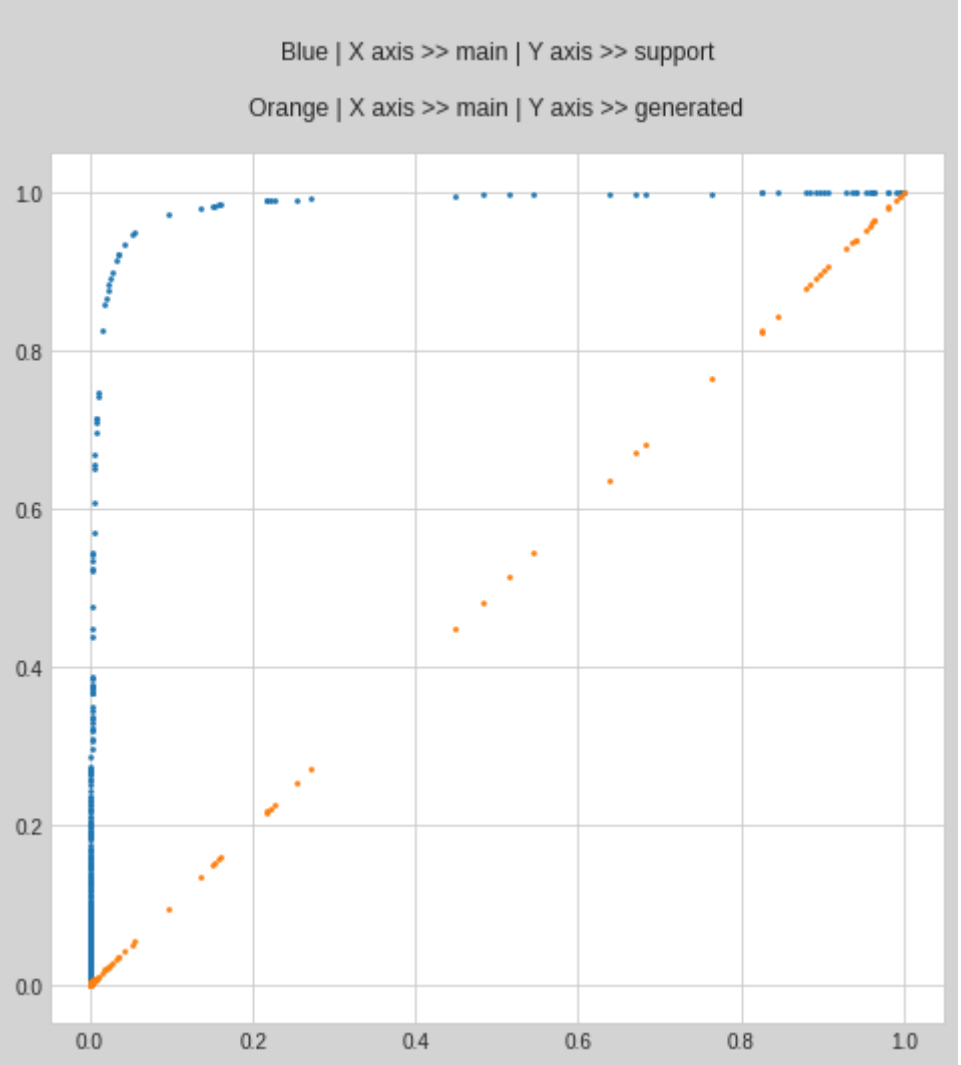
In [28]:
```
drawing1(sub968, sub961, d, 16)
```



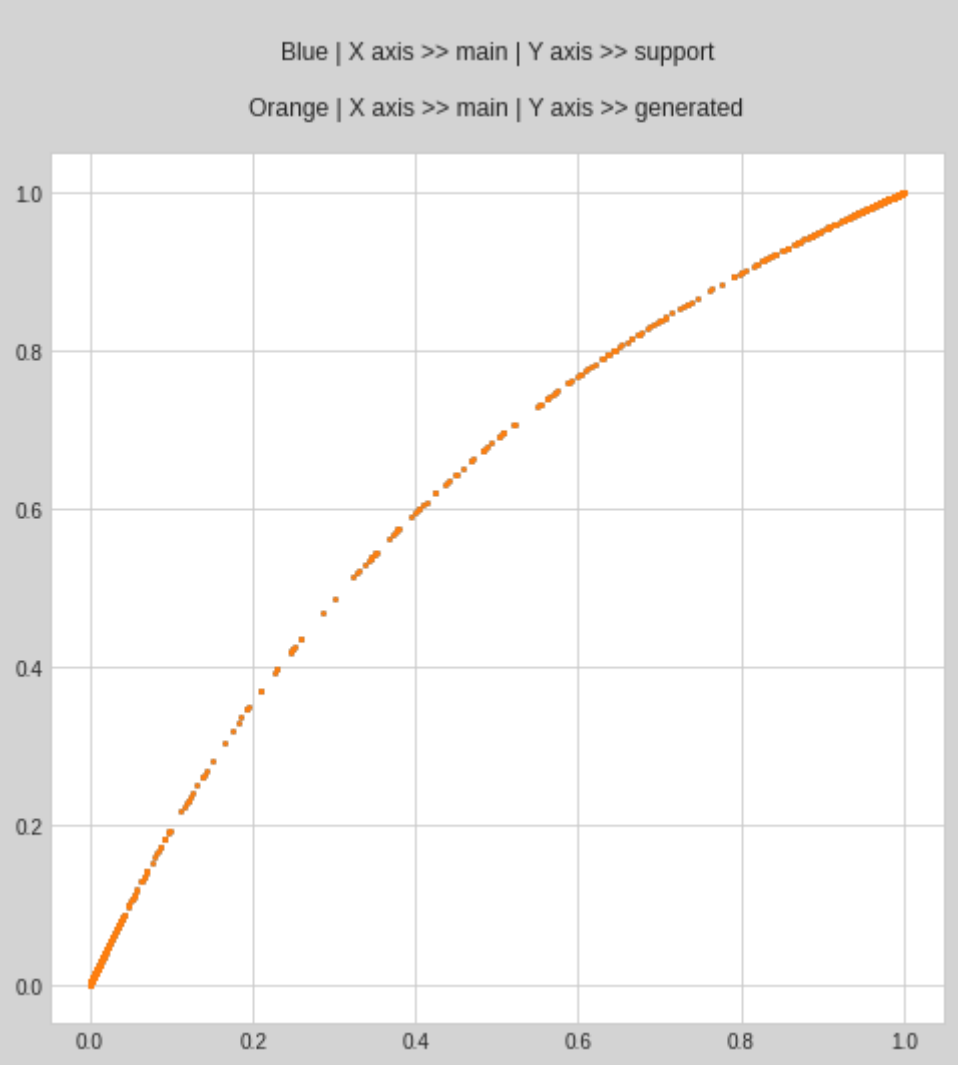
In [29]:
```
drawing1(sub968, sub961, d, 17)
```



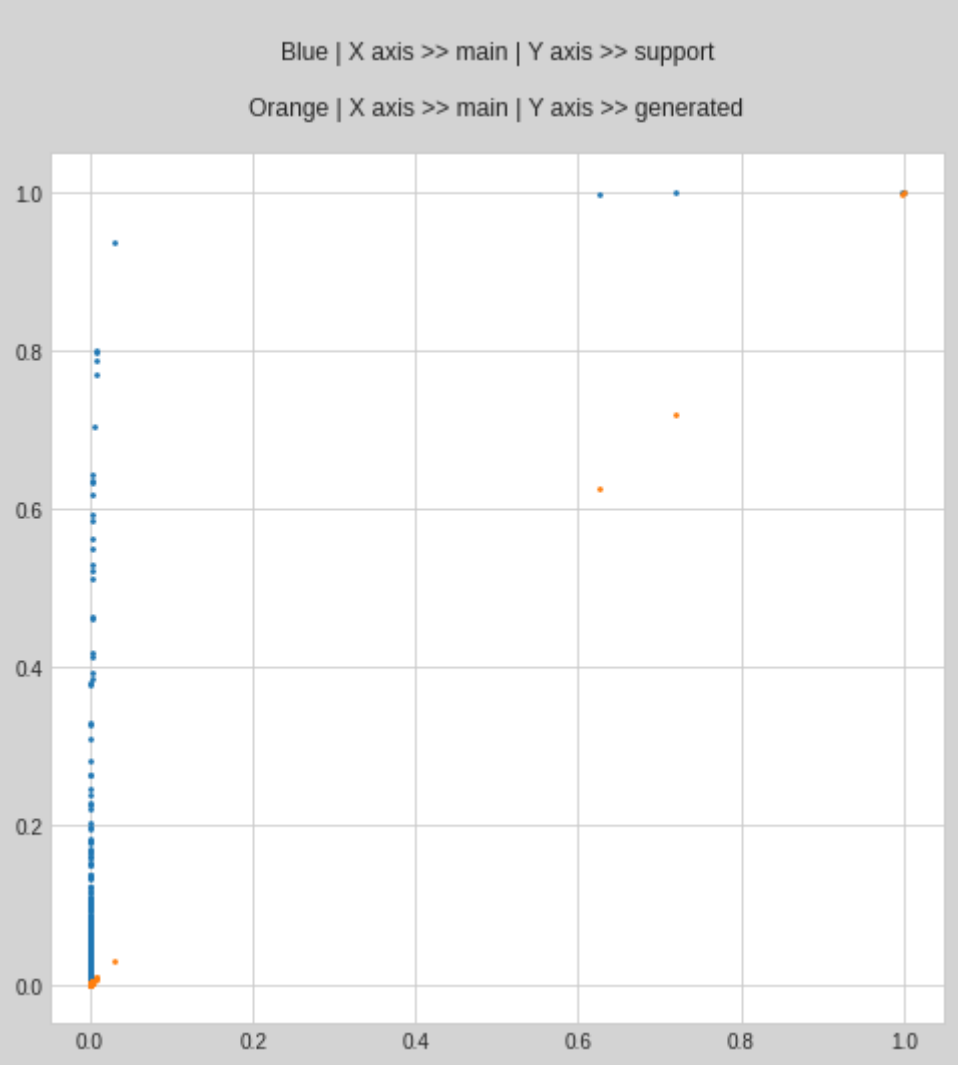
In [30]:
```
drawing1(sub968, sub961, d, 18)
```



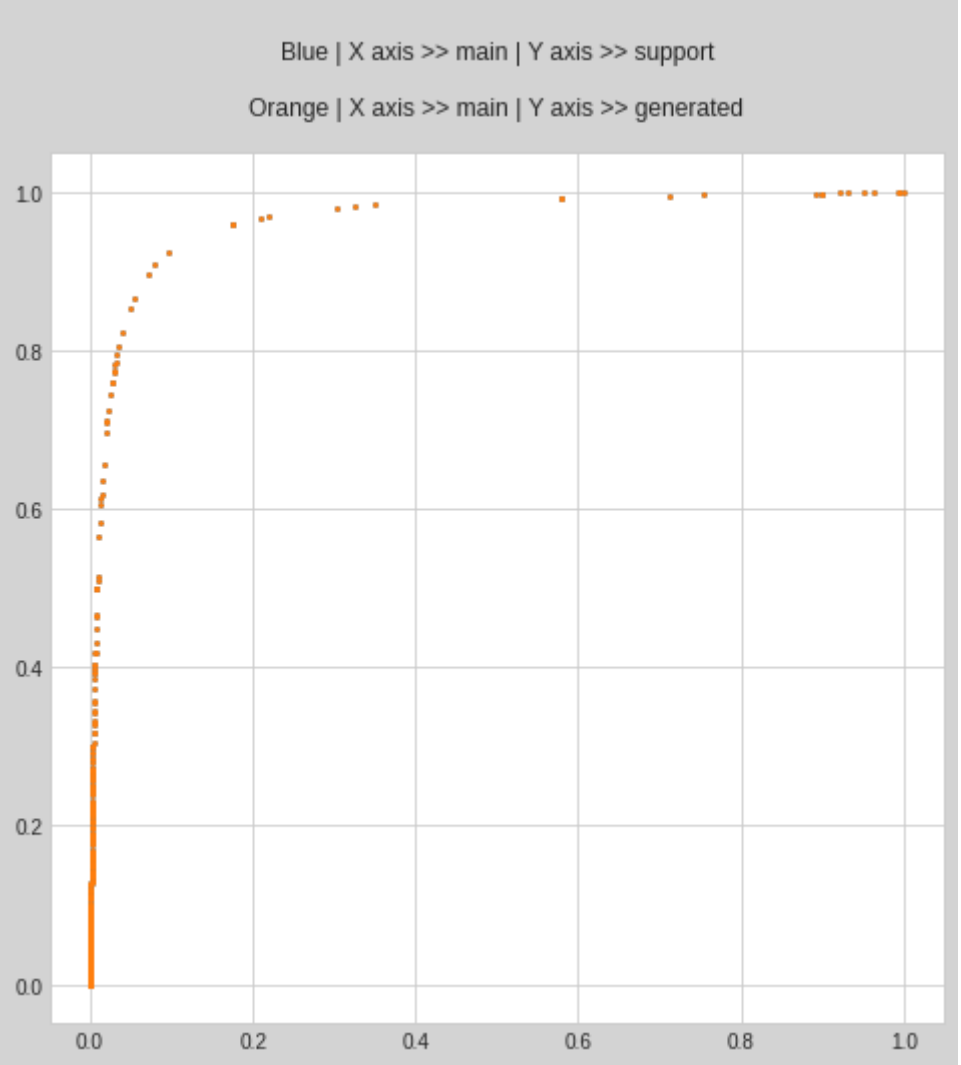
In [31]:
```
drawing1(sub968, sub961, d, 19)
```



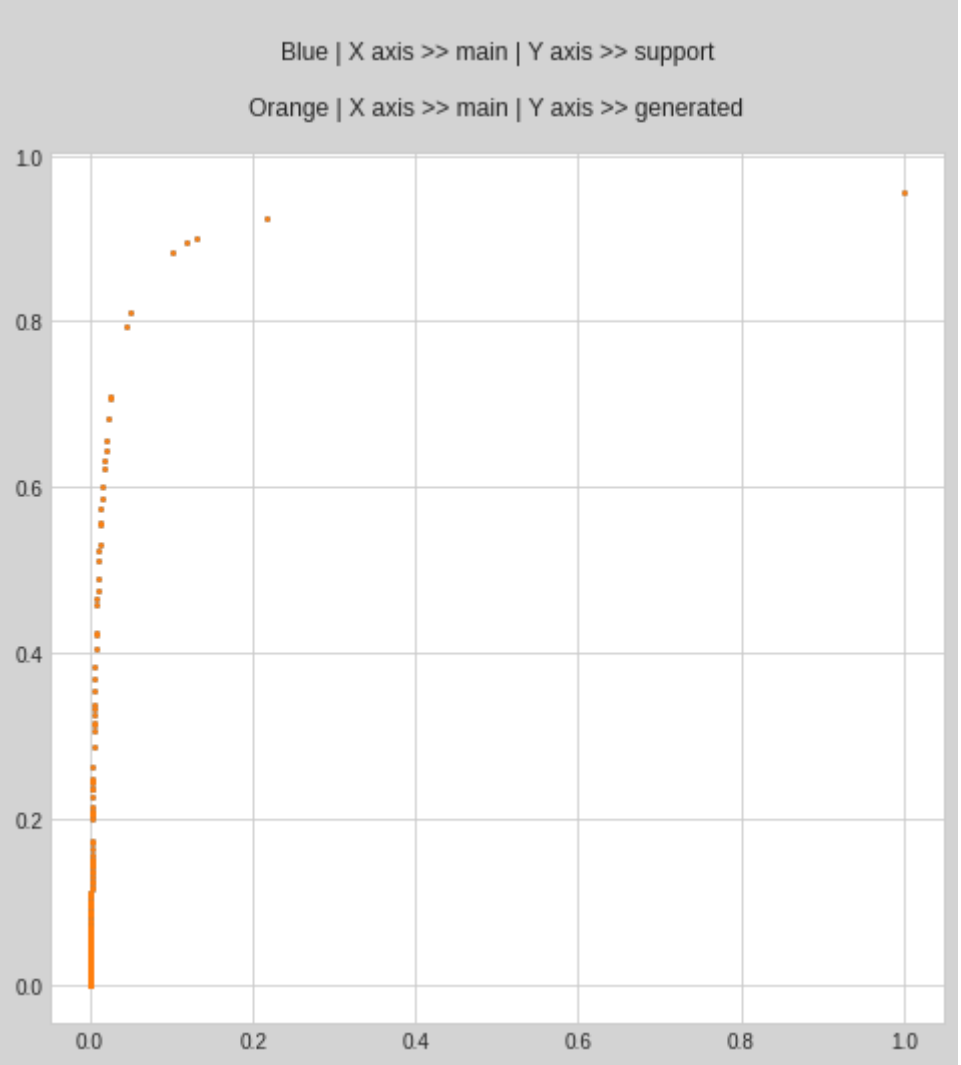
In [32]:
```
drawing1(sub968, sub961, d, 20)
```



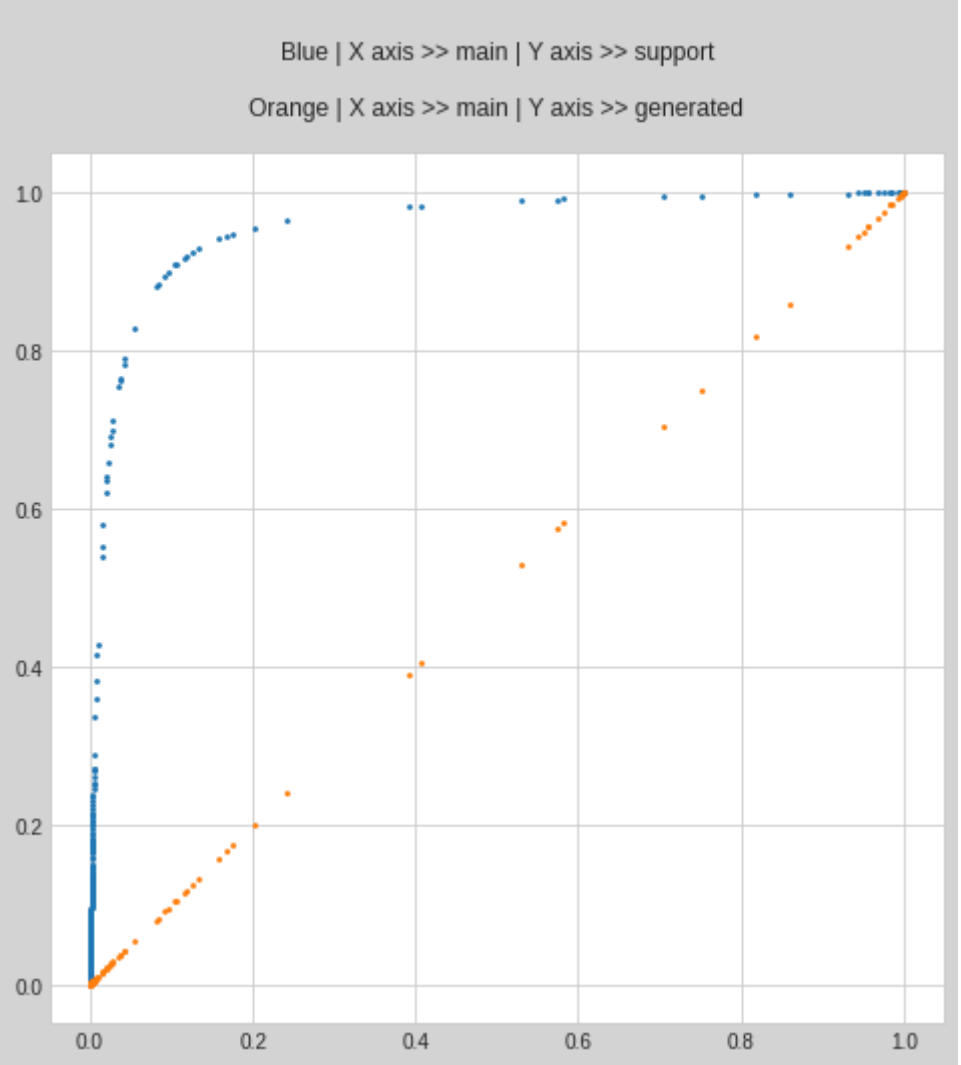
In [33]:
```
drawing1(sub968, sub961, d, 21)
```



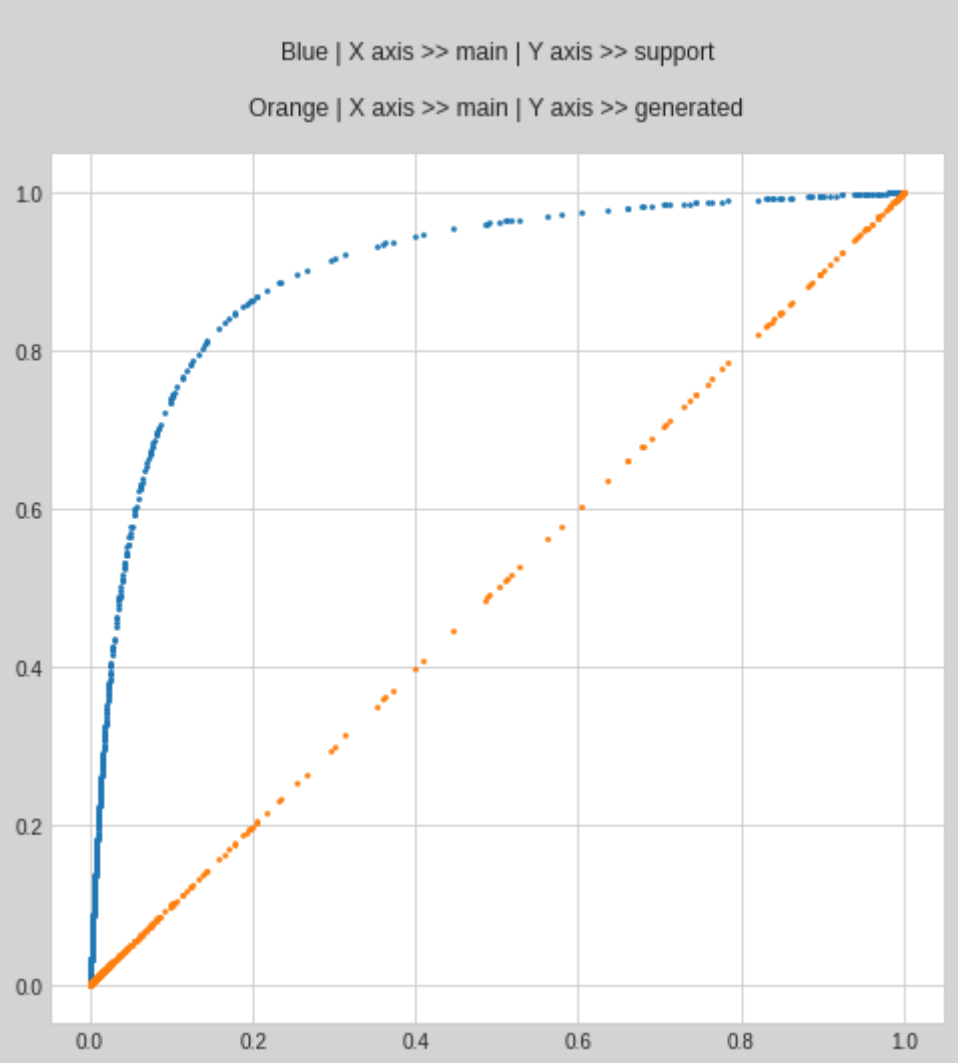
In [34]:
```
drawing1(sub968, sub961, d, 22)
```



In [35]:
```
drawing1(sub968, sub961, d, 23)
```



In [36]:
```
drawing1(sub968, sub961, d, 24)
```



Result

[(Score: 0.968) , (Score: 0.961)] >>> d

d : [(Private Score: 0.97915) , (Public Score: 0.97373)]

In [37]:
```
d.describe()
```

Out[37]:

	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1.992000
mean	0.148069	0.255927	0.058836	0.975920	0.083605	0.049287	3.443965
std	0.324055	0.394491	0.221785	0.095970	0.221616	0.195465	5.238256
min	0.000001	0.000031	0.000000	0.000000	0.000003	0.000000	0.000000
25%	0.000264	0.001760	0.000010	0.997896	0.000638	0.000032	5.145849
50%	0.002151	0.015521	0.000070	0.999792	0.003190	0.000219	4.882412
75%	0.038402	0.411339	0.000732	0.999958	0.023652	0.001828	2.530490
max	0.999999	0.999999	1.000000	1.000000	0.999976	1.000000	1.000000

8 rows × 24 columns

In [38]:
```
# print(d.mean() , sub961.mean())

n1 = d.mean() + sub961.mean()

n1mean = n1.mean()

n2 = n1 / n1mean

n2
```

Out[38]:
```
s0     0.892402
s1     1.542458
s2     0.515516
s3     5.776221
s4     0.503880
s5     0.530050
s6     0.082759
s7     1.837864
s8     0.490991
s9     0.331072
s10    0.252530
s11    1.115764
s12    2.325548
s13    0.253618
s14    0.772764
s15    1.967539
s16    0.437821
s17    0.205713
s18    2.794711
s19    0.082145
s20    0.276296
s21    0.116289
s22    0.192735
s23    0.703314
dtype: float64
```

In [39]:
```
n3 = n2.copy()
for k in range(24):
    n3[k] = 0.95

# n3
```

Result

[(Private Score: 0.97975) , (Public Score: 0.97444)]

In [40]:
```
n4 = n3.copy()

n4[2]   = 0.80

n4[6]   = 0.80

n4[19]  = 0.70

n4[22]  = 0.80

n4[23]  = 0.80

n4
```

Out[40]:
```
s0     0.95
s1     0.95
s2     0.80
s3     0.95
s4     0.95
s5     0.95
s6     0.80
s7     0.95
s8     0.95
s9     0.95
s10    0.95
s11    0.95
s12    0.95
s13    0.95
s14    0.95
s15    0.95
s16    0.95
s17    0.95
s18    0.95
s19    0.70
s20    0.95
s21    0.95
s22    0.80
s23    0.80
dtype: float64
```

In [41]:
```
e = generate1(d, sub961, n4)
```

Result

[(Score: 0.973) , (Score: 0.961)] >>> e

e : [(Private Score: 0.98022) , (Public Score: 0.97490)]

In [42]:
```
e.describe()
```

Out[42]:

	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1.992000e+03	1992.000000	1992.000000	1.992000e+03	1.9920
			54e-02	0.974168	0.083605	5.315334e-02	7.5589

	s0	s1	s2	s3	s4	s5	
std	0.324055	0.394491	2.251883e-01	0.098811	0.221616	1.964352e-01	5.6354
min	0.000001	0.000031	1.379217e-07	0.001901	0.000003	1.763993e-07	4.6538
25%	0.000264	0.001760	9.586658e-05	0.997401	0.000638	1.063986e-04	6.9256
50%	0.002151	0.015521	6.602089e-04	0.999742	0.003190	7.149753e-04	6.5128
75%	0.038402	0.411339	6.751331e-03	0.999947	0.023652	5.682613e-03	3.3288
max	0.999999	0.999999	1.000000e+00	1.000000	0.999976	9.999998e-01	9.9999

8 rows × 24 columns

Submission

In [43]:
```
sub = e
sub.to_csv("submission.csv", index=False)

d.to_csv("submission1.csv", index=False)
e.to_csv("submission2.csv", index=False)

!ls
```

```
__notebook__.ipynb  submission.csv  submission1.csv  submission2.csv
```

ResNeSt: Split-Attention Networks

Hang Zhang[1], Chongruo Wu[2], Zhongyue Zhang[3], Yi Zhu[4], Haibin Lin[5], Zhi Zhang[4], Yue Sun[6], Tong He[4], Jonas Mueller[4], R. Manmatha[4], Mu Li[4], Alexander Smola[4]

Facebook[1], UC Davis[2], Snap[3], Amazon[4], ByteDance[5], SenseTime[6]

zhanghang@fb.com, crwu@ucdavis.edu, zzhang5@snapchat.com, haibin.lin@bytedance.com, sunyue1@sensetime.com, {yzaws,zhiz,htong,jonasmue,manmatha,mli,smola}@amazon.com

Abstract

It is well known that featuremap attention and multi-path representation are important for visual recognition. In this paper, we present a modularized architecture, which applies the channel-wise attention on different network branches to leverage their success in capturing cross-feature interactions and learning diverse representations. Our design results in a simple and unified computation block, which can be parameterized using only a few variables. Our model, named ResNeSt, outperforms EfficientNet in accuracy and latency trade-off on image classification. In addition, ResNeSt has achieved superior transfer learning results on several public benchmarks serving as the backbone, and has been adopted by the winning entries of COCO-LVIS challenge. The source code for complete system and pretrained models are publicly available.

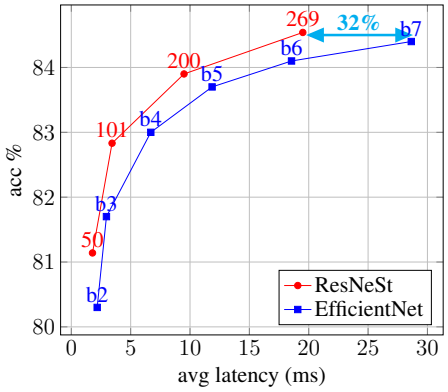


Figure 1: ResNeSt outperforms EfficientNet in accuracy-latency trade-offs on GPU. Notably, ResNeSt-269 has achieved better accuracy than EfficientNet-B7 with 32% less latency. (details in Section 5).

1. Introduction

Deep convolutional neural networks (CNNs) have become the fundamental approach for image classification and other transfer learning tasks in computer vision. As the key component of the CNNs, a convolutional layer learns a set of filters which aggregates the neighborhood information with spatial and channel connections. This operation is suitable to capture *correlated features* with the output channels densely connected to each input channel. Inception models [53, 54] explore the multi-path representation to learn *independent features*, where the input is split into a few lower dimensional embeddings, transformed by different sets of convolutional filters and then merged by concatenation. This strategy encourages the feature exploration by decoupling the input channel connections [63].

The neuron connections in visual cortex have inspired the development of CNNs in the past decades [30]. The main theme of visual representation learning is discovering salient features for a given task [74]. Prior work has modeled spatial and channel dependencies [2, 27, 43], and incorporated attention mechanism [27, 36, 58]. SE-like channel-wise attention [27] employs global pooling to squeeze the channel statistics, and predicts a set of attention factors to apply channel-wise multiplication with the original featuremaps. This mechanism models the interdependencies of featuremap channels, which uses the global context information to selectively highlight or de-emphasize the features [27, 36]. This attention mechanism is similar to attentional selection stage of human primary visual cortex [73], which finds the informative parts for recognizing objects. Human/animals perceive various visual patterns using the cortex in separate regions that respond to different and particular visual features [45]. This strategy makes it easy to identify subtle but dominant differences of similar objects in the neural perception system. Similarly, if we can build a CNN architecture to capture individual salient attributes

for different visual features, we would improve the network representation for image classification.

In this paper, we present a simple architecture which combines the channel-wise attention strategy with multi-path network layout. Our method captures cross-channel feature correlations, while preserving independent representation in the meta structure. A module of our network performs a set of transformations on low dimensional embeddings and concatenates their outputs as in a multi-path network. Each transformation incorporates channel-wise attention strategy to capture interdependencies of the featuremap. We further simplify the architecture to make each transformation share the same topology (*e.g.* Fig 2 (Right)). We can parameterize the network architecture with only a few variables. In addition, such setting also allows us to accelerate the training using identical implementation with unified CNN operators. We refer to such computation block as *Split-Attention Block*. Stacking several Split-Attention blocks in ResNet style, we create a new ResNet variant which we refer to as *Split-Attention Network (ResNeSt)*.

We benchmark the performance of the proposed ResNeSt networks on ImageNet dataset [14]. The proposed ResNeSt achieves better speed-accuracy trade-offs than state-of-the-art CNN models produced via neural architecture search [56] as shown in Table 2. In addition, we also study the transfer learning results on object detection, instance segmentation and semantic segmentation. The proposed ResNeSt has achieved superior performance on several gold-standard benchmarks when serving as the backbone network. For example, our Cascade-RCNN [5] model with ResNeSt-101 backbone achieves 48.3% box mAP and 41.56% mask mAP on MS-COCO instance segmentation. Our DeepLabV3 [9] model, again using a ResNeSt-101 backbone, achieves mIoU of 46.9% on the ADE20K scene parsing validation set, which surpasses the previous best result by more than 1% mIoU. Furthermore, ResNeSt has been adopted by the winning entries of 2020 COCO-LVIS challenge [21, 55, 57].

2. Related Work

CNN Architectures. Since AlexNet [33], deep convolutional neural networks [34] have dominated image classification. With this trend, research has shifted from engineering handcrafted features to engineering network architectures. NIN [38] first uses a global average pooling layer to replace the heavy fully connected layers, and adopts 1×1 convolutional layers to learn non-linear combination of the featuremap channels, which is the first kind of featuremap attention mechanism. VGG-Net [48] proposes a modular network design strategy, stacking the same type of network blocks repeatedly, which simplifies both the workflow of network design and transfer learning for downstream applications. Highway network [51] introduces highway connections which makes the information flow across several layers without attenuation and helps the network convergence. Built on the success of the pioneering work, ResNet [23] introduces an identity skip connection which alleviates the difficulty of vanishing gradient in deep neural network and allows network to learn improved feature representations. ResNet has become one of the most successful CNN architectures which has been adopted in various computer vision applications.

Multi-path and featuremap Attention. Multi-path representation has shown success in GoogleNet [53], in which each network block consists of different convolutional kernels. ResNeXt [64] adopts group convolution [33] in the ResNet bottle block, which converts the multi-path structure into a unified operation. SE-Net [27] introduces a channel-attention mechanism by adaptively recalibrating the channel feature responses. Recently, SK-Net [36] brings the featuremap attention across two network branches. Inspired by the previous methods, our network integrates the channel-wise attention with multi-path network representation.

Neural Architecture Search. With increasing computational power, research interest has begun shifting from manually designed architectures to systematically searched architectures. Recent work explored efficient neural architecture search via parameter sharing [41, 44] and have achieved great success in low-latency and low-complexity CNN models [3, 59]. However, searching a large-scale neural network is still challenging due to the high GPU memory usage via parameter sharing with other architectures. EfficientNet [56] first searches in a small setting and then scale up the network complexity systematically. Instead, we build our model with ResNet meta architecture to scale up the network to deeper versions (from 50 to 269 layers). Our approach also augments the search spaces for neural architecture search and potentially improve the overall performance, which can be studied in the future work.

3. Split-Attention Networks

We now introduce the Split-Attention block, which enables featuremap attention across different featuremap groups in Section 3.1. Later, we describe our network instantiation and how to accelerate this architecture via standard CNN operators in Section 3.2.

3.1. Split-Attention Block

Our *Split-Attention* block is a computational unit, consisting of *featuremap group* and *split attention* operations. Figure 2 (Right) depicts an overview of a Split-Attention Block.

Featuremap Group. As in ResNeXt blocks [64], the feature can be divided into several groups, and the number of featuremap groups is given by a *cardinality* hyperparameter

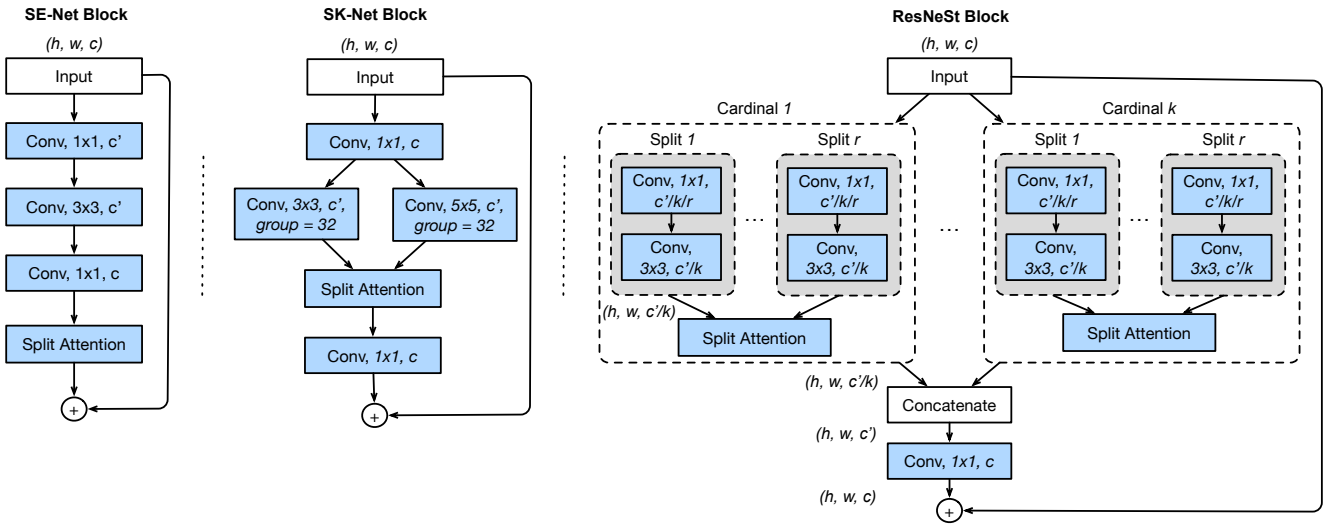


Figure 2: Comparing our ResNeSt block with SE-Net [28] and SK-Net [36]. A detailed view of Split-Attention unit is shown in Figure 3. For simplicity, we show ResNeSt block in cardinality-major view (the featuremap groups with same cardinal group index reside next to each other). We use radix-major in the real implementation, which can be modularized and accelerated by group convolution and standard CNN layers (see supplementary material).

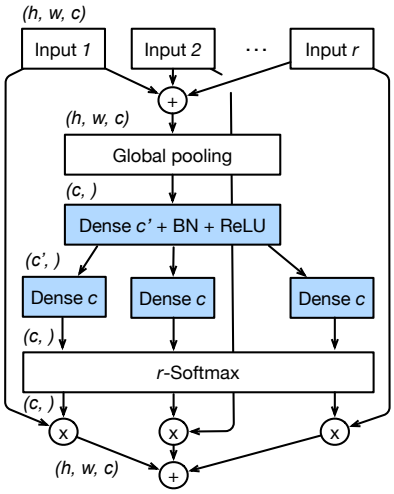


Figure 3: Split-Attention within a cardinal group. For easy visualization in the figure, we use $c = C/K$ in this figure.

K. We refer to the resulting featuremap groups as *cardinal groups*. In this paper, we introduce a new *radix* hyperparameter R that indicates the number of splits within a cardinal group, so the total number of feature groups is $G = KR$. We may apply a series of transformations $\{\mathcal{F}_1, \mathcal{F}_2, ...\mathcal{F}_G\}$ to each individual group, then the intermediate representation of each group is $U_i = \mathcal{F}_i(X)$, for $i \in \{1, 2, ...G\}$.

Split Attention in Cardinal Groups. Following [28, 36], a combined representation for each cardinal group can be obtained by fusing via an element-wise summation across multiple splits. The representation for k-th cardinal group is $\hat{U}^k = \sum_{j=R(k-1)+1}^{Rk} U_j$, where $\hat{U}^k \in \mathbb{R}^{H \times W \times C/K}$ for $k \in 1, 2, ...K$, and H, W and C are the block output featuremap sizes. Global contextual information with embedded channel-wise statistics can be gathered with global average pooling across spatial dimensions $s^k \in \mathbb{R}^{C/K}$ [27, 36]. Here the c-th component is calculated as:

$$s_c^k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \hat{U}_c^k(i, j). \tag{1}$$

A weighted fusion of the cardinal group representation $V^k \in \mathbb{R}^{H \times W \times C/K}$ is aggregated using channel-wise soft attention, where each featuremap channel is produced using a weighted combination over splits. Then the c-th channel is calculated as:

$$V_c^k = \sum_{i=1}^{R} a_i^k(c) U_{R(k-1)+i}, \tag{2}$$

where $a_i^k(c)$ denotes a (soft) assignment weight given by:

$$a_i^k(c) = \begin{cases} \frac{exp(\mathcal{G}_i^c(s^k))}{\sum_{j=1}^{R} exp(\mathcal{G}_j^c(s^k))} & \text{if } R > 1, \\ \frac{1}{1+exp(-\mathcal{G}_i^c(s^k))} & \text{if } R = 1, \end{cases} \tag{3}$$

and mapping $\mathcal{G}_i^c$ determines the weight of each split for the c-th channel based on the global context representation s^k.

ResNeSt Block. The cardinal group representations are then concatenated along the channel dimension: $V = Concat\{V^1, V^2, ...V^K\}$. As in standard residual blocks, the final output Y of our Split-Attention block is produced

using a shortcut connection: $Y = V + X$, if the input and output featuremap share the same shape. For blocks with a stride, an appropriate transformation $\mathcal{T}$ is applied to the shortcut connection to align the output shapes: $Y = V + \mathcal{T}(X)$. For example, $\mathcal{T}$ can be strided convolution or combined convolution-with-pooling.

Instantiation and Computational Costs. Figure 2 (right) shows an instantiation of our Split-Attention block, in which the group transformation $\mathcal{F}_i$ is a 1×1 convolution followed by a 3×3 convolution, and the attention weight function $\mathcal{G}$ is parameterized using two fully connected layers with ReLU activation. The number of parameters and FLOPS of a Split-Attention block are roughly the same as a standard residual block [23, 63] with the same cardinality and number of channels.

Relation to Existing Attention Methods. First introduced in SE-Net [27], the idea of squeeze-and-attention (called *excitation* in the original paper) is to employ a global context to predict channel-wise attention factors. With radix $= 1$, our Split-Attention block is applying a squeeze-and-attention operation to each cardinal group, while the SE-Net operates on top of the entire block regardless of multiple groups. SK-Net [36] introduces feature attention between two network streams. Setting radix $= 2$, the Split-Attention block applies SK-like attention to each cardinal group. Our method generalizes prior work of featuremap attention [27, 36] within a cardinal group setting [63], and its implementation remains computationally efficient. Figure 2 shows an overall comparison with SE-Net and SK-Net blocks.

3.2. Efficient Radix-major Implementation

We refer to the layout described in the previous section as *cardinality-major implementation*, where the featuremap groups with the same cardinal index reside next to each other physically (Figure 2 (Right)). The cardinality-major implementation is straightforward and intuitive, but is difficult to modularize and accelerate using standard CNN operators. For this, we introduce an equivalent *radix-major implementation*.

Figure 4 gives an overview of the Split-Attention block in radix-major layout. The input featuremap is first divided into RK groups, in which each group has a cardinality-index and radix-index. In this layout, the groups with same radix-index reside next to each other. Then, we can conduct a summation across different splits, so that the featuremap groups with the same cardinality-index but different radix-index are fused together. A global pooling layer aggregates over the spatial dimension, while keeps the channel dimension separated, which is identical to conducting global pooling to each individual cardinal groups then concatenate the results. Then two consecutive fully connected (FC) layers with number of groups equal to cardinality are added after

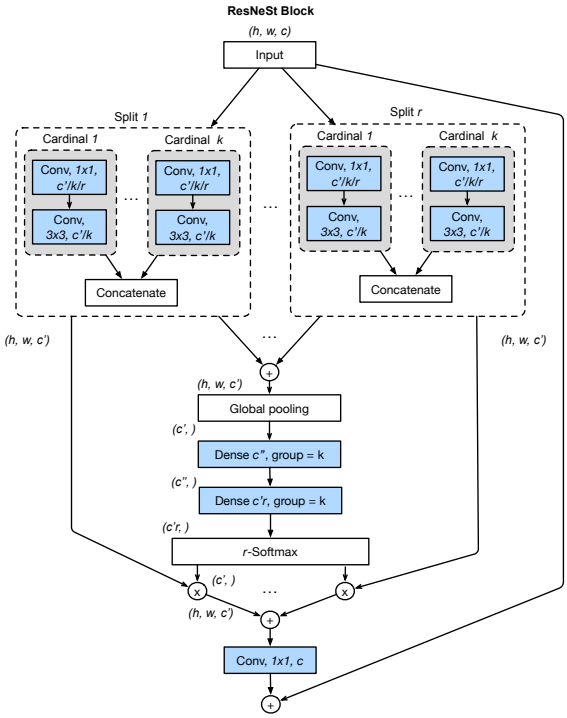


Figure 4: Radix-major implementation of ResNeSt block, where the featuremap groups with same radix index but different cardinality are next to each other physically. This implementation can be implemented using unified CNN operators. (See details in Section 3.2.)

pooling layer to predict the attention weights for each splits. The use of grouped FC layers makes it identical to apply each pair of FCs separately on top each cardinal groups.

With this implementation, the first 1×1 convolutional layers can be unified into one layer and the 3×3 convolutional layers can be implemented using a single grouped convolution with the number of groups of RK. Therefore, the Split-Attention block is modularized using standard CNN operators.

4. Network and Training

We now describe the network design and training strategies used in our experiments. First, we detail a couple of tweaks that further improve performance, some of which have been empirically validated in [25].

4.1. Network Tweaks

Average Downsampling. For transfer learning on dense prediction tasks such as detection or segmentation, it becomes essential to preserve spatial information. Recent ResNet implementations usually apply the strided convolution at the 3×3 layer instead of the previous 1×1 layer to better preserve such information [26, 28]. Convolutional

layers require handling featuremap boundaries with zero-padding strategies, which is often suboptimal when transferring to other dense prediction tasks. Instead of using strided convolution at the transitioning block (in which the spatial resolution is downsampled), we use an average pooling layer with a kernel size of 3×3 .

Tweaks from ResNet-D. We also adopt two simple yet effective ResNet modifications introduced by [26]: (1) The first 7×7 convolutional layer is replaced with three consecutive 3×3 convolutional layers, which have the same receptive field size with a similar computation cost as the original design. (2) A 2×2 average pooling layer is added to the shortcut connection prior to the 1×1 convolutional layer for the transitioning blocks with stride of two.

4.2. Training Strategy

Large Mini-batch Distributed Training.[1] For effectively training deep CNN models, we follow the prior work [18, 35, 37] to train our models using 8 servers (64 GPUs in total) in parallel. Our learning rates are adjusted according to a cosine schedule [26, 29]. We follow the common practice using linearly scaling-up the initial learning rate based on the mini-batch size. The initial learning rate is given by $\eta = \frac{B}{256}\eta_{base}$, where B is the mini-batch size and we use $\eta_{base} = 0.1$ as the base learning rate. This warm-up strategy is applied over the first 5 epochs, gradually increasing the learning rate linearly from 0 to the initial value for the cosine schedule [18, 37]. The batch normalization (BN) parameter γ is initialized to zero in the final BN operation of each block, as has been suggested for large batch training [18].

Label Smoothing. Label smoothing was first used to improve the training of Inception-V2 [54]. Recall the cross entropy loss incurred by our network's predicted class probabilities q is computed against ground-truth p as:

$$\ell(p, q) = -\sum_{i=1}^{K} p_i \log q_i, \tag{4}$$

where K is total number of classes, p_i is the ground truth probability of the i-th class, and q_i is the network's predicted probability for the i-th class. As in standard image classification, $q_i = \frac{exp(z_i)}{\sum_{j=1}^{K} exp(z_j)}$ where z_i are the logits produced by our network's ouput layer. When the provided labels are classes rather than class-probabilities (hard labels), $p_i = 1$ if i equals the ground truth class c, and is otherwise $= 0$. Thus in this setting: $\ell_{hard}(p, q) = -\log q_c = -z_c + \log(\sum_{j=1}^{K} exp(z_j))$. During the final phase of training, the logits z_j tend to be very small for $j \neq c$, while z_c is being pushed to its optimal value ∞, and this can induce overfitting [26, 54]. Rather than assigning hard labels as targets, label smoothing uses a smoothed ground truth probability:

$$p_i = \begin{cases} 1 - \varepsilon & \text{if } i = c, \\ \varepsilon/(K-1) & \text{otherwise} \end{cases} \tag{5}$$

with small constant $\varepsilon > 0$. This mitigates network overconfidence and overfitting.

Auto Augmentation. Auto-Augment [12] is a strategy that augments the training data with transformed images, where the transformations are learned adaptively. 16 different types of image jittering transformations are introduced, and from these, one augments the data based on 24 different combinations of two consecutive transformations such as shift, rotation, and color jittering. The magnitude of each transformation can be controlled with a relative parameter (e.g. rotation angle), and transformations may be probabilistically skipped.

Mixup Training. Mixup is another data augmentation strategy that generates a weighted combinations of random image pairs from the training data [67]. Given two images and their ground truth labels: $(x^{(i)}, y^{(i)}), (x^{(j)}, y^{(j)})$, a synthetic training example $(\hat{x}, \hat{y})$ is generated as:

$$\hat{x} = \lambda x^i + (1 - \lambda) x^j, \tag{6}$$

$$\hat{y} = \lambda y^i + (1 - \lambda) y^j, \tag{7}$$

where $\lambda \sim \text{Beta}(\alpha = 0.2)$ is independently sampled for each augmented example.

Large Crop Size. Image classification research typically compares the performance of different networks operating on images that share the same crop size. ResNet variants [23, 26, 27, 63] usually use a fixed training crop size of 224, while the Inception-Net family [52–54] uses a training crop size of 299. Recently, the EfficientNet method [56] has demonstrated that increasing the input image size for a deeper and wider network may better trade off accuracy vs. FLOPS. For fair comparison, we use a crop size of 224 when comparing our ResNeSt with ResNet variants, and a crop size of 256 when comparing with other approaches.

Regularization. Very deep neural networks tend to overfit even for large datasets [70]. To prevent this, dropout regularization randomly masks out some neurons during training (but not during inference) to form an implicit network ensemble [27, 50, 70]. A dropout layer with the dropout probability of 0.2 is applied before the final fully-connected layer to the networks with more than 200 layers. We also apply DropBlock layers to the convolutional layers at the last two stages of the network. As a structured variant of dropout, DropBlock [17] randomly masks out local block regions, and is more effective than dropout for specifically regularizing convolutional layers.

Finally, we also apply weight decay (i.e. L2 regularization) which additionally helps stabilize training. We only

[1] Note that large mini-batch training does not improve network accuracy. Instead, it often degrades the results.

	#P	GFLOPs	acc(%)
ResNetD-50 [26]	25.6M	4.34	78.31
+ mixup	25.6M	4.34	79.15
+ autoaug	25.6M	4.34	79.41
ResNeSt-50-fast	27.5M	4.34	80.64
ResNeSt-50	27.5M	5.39	81.13

Variant	#P	GFLOPs	img/sec	acc(%)
0s1x64d	25.6M	4.34	688.2	79.41
1s1x64d	26.3M	4.34	617.6	80.35
2s1x64d	27.5M	4.34	533.0	80.64
4s1x64d	31.9M	4.35	458.3	80.90
2s2x40d	26.9M	4.38	481.8	81.00

Table 1: Ablation study for ImageNet image classification. (Left) breakdown of improvements. (Right) *radix vs. cardinality* under ResNeSt-fast setting. For example *2s2x40d* denotes radix=2, cardinality=2 and width=40. Note that even radix=1 does not degrade any existing approach (see Equation 3).

apply weight decay to the weights of convolutional and fully connected layers [18, 26].

5. Image Classification Results

Our first experiments study the image classification performance of ResNeSt on the ImageNet 2012 dataset [14] with 1.28M training images and 50K validation images (from 1000 different classes). As is standard, networks are trained on the training set and we report their top-1 accuracy on the validation set.

5.1. Implementation Details

We use data sharding for distributed training on ImageNet, evenly partitioning the data across GPUs. At each training iteration, a mini-batch of training data is sampled from the corresponding shard (without replacement). We apply the transformations from the learned Auto Augmentation policy to each individual image. Then we further apply standard transformations including: random size crop, random horizontal flip, color jittering, and changing the lighting. Finally, the image data are RGB-normalized via mean/standard-deviation rescaling. For mixup training, we simply mix each sample from the current mini-batch with its reversed order sample [26]. Batch Normalization [31] is used after each convolutional layer before ReLU activation [42]. Network weights are initialized using Kaiming Initialization [24]. A drop layer is inserted before the final classification layer with dropout ratio $= 0.2$. Training is done for 270 epochs with a weight decay of 0.0001 and momentum of 0.9, using a cosine learning rate schedule with the first 5 epochs reserved for warm-up. We use a mini-batch of size 8192 for ResNeSt-50, 4096 for ResNeSt 101, and 2048 for ResNeSt-$\{200, 269\}$. For evaluation, we first resize each image to 1/0.875 of the crop size along the short edge and apply a center crop. Our code implementation for ImageNet training uses GluonCV [19] with MXNet [10].

5.2. Ablation Study

ResNeSt is based on the ResNet-D model [26]. Mixup training improves the accuracy of ResNetD-50 from 78.31% to 79.15%. Auto augmentation further improves the accuracy by 0.26%. When employing our Split-Attention block to form a *ResNeSt-50-fast* model, accuracy is further boosted to 80.64%. In this ResNeSt-fast setting, the effective average downsampling is applied prior to the 3×3 convolution to avoid introducing extra computational costs in the model. With the downsampling operation moved after the convolutional layer, ResNeSt-50 achieves 81.13% accuracy.

Radix vs. Cardinality. We conduct an ablation study on ResNeSt-variants with different radix/cardinality. In each variant, we adjust the network's width appropriately so that its overall computational cost remains similar to the ResNet variants. The results are shown in Table 1, where s denotes the radix, x the cardinality, and d the network width ($0s$ represents the use of a standard residual block as in ResNet-D [26]). We empirically find that increasing the radix from 0 to 4 continuously improves the top-1 accuracy, while also increasing latency and memory usage. Although we expect further accuracy improvements with even greater radix/cardinality, we employ Split-Attention with the $2s1x64d$ setting in subsequent experiments, to ensure these blocks scale to deeper networks with a good trade-off between speed, accuracy and memory usage.

5.3. Comparing against the State-of-the-Art

To compare with CNN models trained using different crop size settings, we increase the training crop size for deeper models. We use a crop size of 256×256 for ResNeSt-200 and 320×320 for ResNeSt-269. Bicubic upsampling strategy is employed for input-size greater than 256. The results are shown in Table 2, where we compare the inference speed in addition to the number of parameters. We find that despite its advantage in parameters with accuracy trade-off, the widely used depth-wise convolution is not optimized for inference speed. In this benchmark, all inference speeds are measured using a mini-batch of 16 using the implementation [1] from the original author on a single NVIDIA V100 GPU. The proposed ResNeSt has better accuracy and latency trade-off than models found via neural architecture search.

	#P	crop	img/sec	acc(%)
ResNeSt-101(ours)	48M	256	**291.3**	**83.0**
EfficientNet-B4 [56]	19M	380	149.3	83.0
SENet-154 [27]	146M	320	133.8	82.7
NASNet-A [78]	89M	331	103.3	82.7
AmoebaNet-A [46]	87M	299	-	82.8
ResNeSt-200 (ours)	70M	320	**105.3**	**83.9**
EfficientNet-B5 [56]	30M	456	84.3	83.7
AmoebaNet-C [46]	155M	299	-	83.5
ResNeSt-269 (ours)	111M	416	**51.2**	**84.5**
GPipe	557M	-	-	84.3
EfficientNet-B7 [56]	66M	600	34.9	84.4

Table 2: Accuracy vs. Throughput for SoTA CNN models on ImageNet. Our ResNeSt model displays the best trade-off. Average Inference latency is measured on a NVIDIA V100 GPU using the original code implementation of each model with a mini-batch of size 16.

	Method	Backbone	mAP%
Prior Work	Faster-RCNN [47]	ResNet101 [22]	37.3
		ResNeXt101 [7, 63]	40.1
		SE-ResNet101 [27]	41.9
	Faster-RCNN+DCN [13]	ResNet101 [7]	42.1
	Cascade-RCNN [4]	ResNet101	42.8
Our Results	Faster-RCNN [47]	ResNet50 [60]	39.25
		ResNet101 [60]	41.37
		ResNeSt50 (ours)	42.33
		ResNeSt101 (ours)	**44.72**
	Cascade-RCNN [4]	ResNet50 [60]	42.52
		ResNet101 [60]	44.03
		ResNeSt50 (ours)	45.41
		ResNeSt101 (ours)	**47.50**
	Cascade-RCNN [4]	ResNeSt200 (ours)	49.03

Table 3: Object detection results on the MS-COCO validation set. Both Faster-RCNN and Cascade-RCNN are significantly improved by our ResNeSt backbone.

6. Transfer Learning Results

6.1. Object Detection

We report our detection result on MS-COCO [40] in Table 9. All models are trained on COCO-2017 training set with 118k images, and evaluated on COCO-2017 validation set with 5k images (aka. minival) using the standard COCO AP metric of single scale. We train all models with FPN [39], synchronized batch normalization [68] and image scale augmentation (short size of a image is picked randomly from 640 to 800). 1x learning rate schedule is used. We conduct Faster-RCNNs and Cascade-RCNNs experiments using Detectron2 [60]. For comparison, we simply replaced the vanilla ResNet backbones with our ResNeSt, while using the default settings for the hyper-parameters and detection heads [20, 60].

Compared to the baselines using standard ResNet, Our backbone is able to boost mean average precision by around 3% on both Faster-RCNNs and Cascade-RCNNs. The result demonstrates our backbone has good generalization ability and can be easily transferred to the downstream task. Notably, our ResNeSt50 outperforms ResNet101 on both Faster-RCNN and Cascade-RCNN detection models, using significantly fewer parameters. Detailed results in Table 9. We evaluate our Cascade-RCNN with ResNeSt101 deformable, that is trained using 1x learning rate schedule on COCO test-dev set as well. It yields a box mAP of 49.2 using single scale inference.

6.2. Instance Segmentation

To explore the generalization ability of our novel backbone, we also apply it to instance segmentation tasks. Besides the bounding box and category probability, instance segmentation also predicts object masks, for which a more accurate dense image representation is desirable.

We evaluate the Mask-RCNN [22] and Cascade-Mask-RCNN [4] models with ResNeSt-50 and ResNeSt-101 as their backbones. All models are trained along with FPN [39] and synchronized batch normalization. For data augmentation, input images' shorter side are randomly scaled to one of (640, 672, 704, 736, 768, 800). To fairly compare it with other methods, 1x learning rate schedule policy is applied, and other hyper-parameters remain the same. We re-train the baseline with the same setting described above, but with the standard ResNet. All our experiments are trained on COCO-2017 dataset and using Detectron2 [60]. For the baseline experiments, the backbone we used by default is the MSRA version of ResNet, having stride-2 on the 1x1 conv layer. Both bounding box and mask mAP are reported on COCO-2017 validation dataset.

As shown in Table 4, our new backbone achieves better performance. For Mask-RCNN, ResNeSt50 outperforms the baseline with a gain of 2.85%/2.09% for box/mask performance, and ResNeSt101 exhibits even better improvement of 4.03%/3.14%. For Cascade-Mask-RCNN, the gains produced by switching to ResNeSt50 or ResNeSt101 are 3.13%/2.36% or 3.51%/3.04%, respectively. This suggests a model will be better if it consists of more Split-Attention modules. As observed in the detection results, the mAP of our ResNeSt50 exceeds the result of the standard ResNet101 backbone, which indicates a higher capacity of the small model with our proposed module. Finally, we also train a Cascade-Mask-RCNN with ResNeSt101-deformable using a 1x learning rate schedule. We evaluate it on the COCO test-dev set, yielding 50.0 box mAP, and 43.1 mask mAP respectively. Additional experiments under different settings are included in the supplementary material.

	Method	Backbone	box mAP%	mask mAP%
Prior Work	DCV-V2 [76]	ResNet50	42.7	37.0
	HTC [6]	ResNet50	43.2	38.0
	Mask-RCNN [22]	ResNet101 [7]	39.9	36.1
	Cascade-RCNN [5]	ResNet101	44.8	38.0
Our Results	Mask-RCNN [22]	ResNet50 [60]	39.97	36.05
		ResNet101 [60]	41.78	37.51
		ResNeSt50 (ours)	42.81	38.14
		ResNeSt101 (ours)	**45.75**	**40.65**
	Cascade-RCNN [4]	ResNet50 [60]	43.06	37.19
		ResNet101 [60]	44.79	38.52
		ResNeSt50 (ours)	46.19	39.55
		ResNeSt101 (ours)	**48.30**	**41.56**

Table 4: Instance Segmentation results on the MS-COCO validation set. Both Mask-RCNN and Cascade-RCNN models are improved by our ResNeSt backbone. Models with our ResNeSt-101 outperform all prior work using ResNet-101.

6.3. Semantic Segmentation

In transfer learning for semantic segmentation, we use the GluonCV [19] implementation of DeepLabV3 [9] as a baseline approach. Here a dilated network strategy [8, 65] is applied to the backbone network, resulting in a stride-8 model. Synchronized Batch Normalization [68] is used during training, along with a polynomial-like learning rate schedule (with initial learning rate $= 0.1$). For evaluation, the network prediction logits are upsampled 8 times to calculate the per-pixel cross entropy loss against the ground truth labels. We use multi-scale evaluation with flipping [68, 71, 77].

We first consider the Cityscapes [11] dataset, which consists of 5K high-quality labeled images. We train each model on 2,975 images from the training set and report its mIoU on 500 validation images. Following prior work, we only consider 19 object/stuff categories in this benchmark. We have not used any coarse labeled images or any extra data in this benchmark. Our ResNeSt backbone boosts the mIoU achieved by DeepLabV3 models by around 1% while maintaining a similar overall model complexity. Notably, the DeepLabV3 model using our ResNeSt-50 backbone already achieves better performance than DeepLabV3 with a much larger ResNet-101 backbone.

ADE20K [75] is a large scene parsing dataset with 150 object and stuff classes containing 20K training, 2K validation, and 3K test images. All networks are trained on the training set for 120 epochs and evaluated on the validation set. Table 6 shows the resulting pixel accuracy (pixAcc) and mean intersection-of-union (mIoU). The performance of the DeepLabV3 models are dramatically improved by employing our ResNeSt backbone. Analogous to previous results, the DeepLabv3 model using our ResNeSt-50 backbone already outperforms DeepLabv3 using a deeper ResNet-101 backbone. DeepLabV3 with a ResNeSt-101 backbone achieves 82.07% pixAcc and 46.91% mIoU, which to our knowledge, is the best single model that has been presented for ADE20K.

	Method	Backbone	pixAcc%	mIoU%
Prior Work	UperNet [62]	ResNet101	81.01	42.66
	PSPNet [71]	ResNet101	81.39	43.29
	EncNet [68]	ResNet101	81.69	44.65
	CFNet [69]	ResNet101	81.57	44.89
	OCNet [66]	ResNet101	-	45.45
	ACNet [16]	ResNet101	81.96	45.90
Ours	DeeplabV3 [9]	ResNet50 [19]	80.39	42.1
		ResNet101 [19]	81.11	44.14
		ResNeSt-50 (ours)	81.17	45.12
		ResNeSt-101 (ours)	**82.07**	**46.91**
		ResNeSt-200 (ours)	82.45	48.36

Table 5: Semantic segmentation results on validation set of: ADE20K.

	Method	Backbone	mIoU%
Prior Work	DANet [15]	ResNet101	77.6
	PSANet [72]	ResNet101	77.9
	PSPNet [71]	ResNet101	78.4
	AAF [32]	ResNet101	79.2
	DeeplabV3 [9]	ResNet101	79.3
	OCNet [66]	ResNet101	80.1
Ours	DeeplabV3 [9]	ResNet50 [19]	78.72
		ResNet101 [19]	79.42
		ResNeSt-50 (ours)	79.87
		ResNeSt-101 (ours)	**80.42**
		ResNeSt-200 (ours)	82.7

Table 6: Semantic segmentation results on validation set of Citscapes. Models are trained without coarse labels or extra data.

7. Conclusion

This work proposes the ResNeSt architecture that leverages the channel-wise attention with multi-path representation into a single unified Split-Attention block. The model universally improves the learned feature representations to boost performance across image classification, object detection, instance segmentation and semantic segmentation. Our Split-Attention block is easy to work with (i.e., drop-in replacement of a standard residual block), computationally efficient (i.e., 32% less latency than EfficientNet-B7 but with better accuracy), and transfers well. We believe ResNeSt can have an impact across multiple vision tasks, as it has already been adopted by multiple winning entries in 2020 COCO-LVIS challenge and 2020 DAVIS-VOS chanllenge.

References

[1] Tensorflow Efficientnet. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet. Accessed: 2020-03-04. 6

[2] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2874–2883, 2016. 1

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 2

[4] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 7, 8, 12

[5] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: High quality object detection and instance segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 2, 8, 12

[6] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4974–4983, 2019. 8

[7] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 7, 8

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv:1606.00915*, 2016. 8

[9] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 2, 8

[10] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 6

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 8

[12] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123, 2019. 5

[13] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017. 7, 12

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 2, 6

[15] Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. Dual Attention Network for Scene Segmentation. 2019. 8

[16] Jun Fu, Jing Liu, Yuhang Wang, Yong Li, Yongjun Bao, Jinhui Tang, and Hanqing Lu. Adaptive context network for scene parsing. In *Proceedings of the IEEE international conference on computer vision*, pages 6748–6757, 2019. 8

[17] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10727–10737, 2018. 5

[18] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 5, 6

[19] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, et al. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *Journal of Machine Learning Research*, 21(23):1–7, 2020. 6, 8, 12

[20] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, Shuai Zheng, and Yi Zhu. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *Journal of Machine Learning Research*, 21(23):1–7, 2020. 7

[21] Jianhua Han, Minzhe Niu, Zewei Du, Longhui Wei, Lingxi Xie, Xiaopeng Zhang, and Qi Tian. Asynchronous semi-supervised learning for large vocabulary instance segmentation, 2020. 2

[22] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017. 7, 8, 12

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 2, 4, 5, 12

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 6

[25] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks to train convolutional neural networks for image classification. *arXiv preprint arXiv:1812.01187*, 2018. 4

[26] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 4, 5, 6

[27] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017. 1, 2, 3, 4, 5, 7

[28] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 3, 4

[29] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 5

[30] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962. 1

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 6

[32] Tsung-Wei Ke, Jyh-Jing Hwang, Ziwei Liu, and Stella X. Yu. Adaptive Affinity Fields for Semantic Segmentation. In *European Conference on Computer Vision (ECCV)*, 2018. 8

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2

[35] Mu Li. *Scaling distributed machine learning with system and algorithm co-design*. PhD thesis, PhD thesis, Intel, 2017. 5

[36] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 510–519, 2019. 1, 2, 3, 4

[37] Haibin Lin, Hang Zhang, Yifei Ma, Tong He, Zhi Zhang, Sheng Zha, and Mu Li. Dynamic mini-batch sgd for elastic distributed training: Learning in the limbo of resources. *arXiv preprint arXiv:1904.12043*, 2019. 5

[38] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 2

[39] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 7

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 7

[41] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2

[42] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 6

[43] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016. 1

[44] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 2

[45] Rishi Rajalingham and James J DiCarlo. Reversible inactivation of different millimeter-scale regions of primate it results in different patterns of core object recognition deficits. *Neuron*, 102(2):493–505, 2019. 1

[46] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 7

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 7

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[49] Bharat Singh, Mahyar Najibi, and Larry S Davis. Sniper: Efficient multi-scale training. In *Advances in neural information processing systems*, pages 9310–9320, 2018. 12

[50] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014. 5

[51] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. 2

[52] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017. 5

[53] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 2, 5

[54] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 1, 5

[55] Jingru Tan, Gang Zhang, Hanming Deng, Changbao Wang, Lewei Lu, Quanquan Li, and Jifeng Dai. 1st place solution of lvis challenge 2020: A good box is not a guarantee of a good mask. *arXiv preprint arXiv:2009.01559*, 2020. 2

[56] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 2, 5, 7

[57] Jiaqi Wang, Wenwei Zhang, Yuhang Zang, Yuhang Cao, Jiangmiao Pang, Tao Gong, Kai Chen, Ziwei Liu, Chen Change Loy, and Dahua Lin. Seesaw loss

for long-tailed instance segmentation. *arXiv preprint arXiv:2008.10032*, 2020. 2

[58] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 1

[59] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 2

[60] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 7, 8, 12

[61] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 466–481, 2018. 12

[62] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. *arXiv preprint arXiv:1807.10221*, 2018. 8

[63] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016. 1, 4, 5, 7, 12

[64] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 2

[65] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 8

[66] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. *arXiv preprint arXiv:1909.11065*, 2019. 8

[67] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 5

[68] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 7, 8

[69] Hang Zhang, Han Zhang, Chenguang Wang, and Junyuan Xie. Co-occurrent features in semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 548–557, 2019. 8

[70] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 718–726, 2017. 5

[71] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 8

[72] Hengshuang Zhao, Yi Zhang, Shu Liu, Jianping Shi, Chen Change Loy, Dahua Lin, and Jiaya Jia. PSANet: Point-wise Spatial Attention Network for Scene Parsing. In *European Conference on Computer Vision (ECCV)*, 2018. 8

[73] Li Zhaoping and Zhaoping Li. *Understanding vision: theory, models, and data*. Oxford University Press, USA, 2014. 1

[74] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems*, pages 487–495, 2014. 1

[75] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proc. CVPR*, 2017. 8

[76] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9308–9316, 2019. 8, 12

[77] Yi Zhu, Karan Sapra, Fitsum A. Reda, Kevin J. Shih, Shawn Newsam, Andrew Tao, and Bryan Catanzaro. Improving Semantic Segmentation via Video Propagation and Label Relaxation. 2019. 8

[78] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 7

Method	Backbone	OKS AP% w/o flip	OKS AP% w/ flip
SimplePose [61]	ResNet50 [19]	71.0/91.2/78.6	72.2/92.2/79.9
	ResNet101 [19]	72.6/91.3/80.8	73.6/92.3/81.1
	ResNeSt50 (ours)	72.3/92.3/80.0	73.4/92.4/81.2
	ResNeSt101 (ours)	73.6/92.3/80.9	74.6/92.4/82.1

Table 7: Pose estimation results on MS-COCO dataset in terms of OKS AP.

	Method	Backbone	Deformable	mAP%
Prior Work	DCNv2 [76]	ResNet101 [23]	v2	44.8
		ResNeXt101 [63]	v2	45.3
	TridentNet [13]	ResNet101 [23]	v1	46.8
		ResNet101* [13]	v1	48.4
	SNIPER [49]	ResNet101* [13]	v1	46.1
	Cascade-RCNN [5]	ResNet101 [23]	n/a	42.8
	Cascade-RCNN [5]	ResNeSt101 (ours)	v2	**49.2**

Table 8: Object detection results on the MS-COCO test-dev set. The single model of Cascade-RCNN with ResNeSt backbone using deformable convolution [13] achieves 49.2% mAP, which outperforms all previous methods. (* means using multi-scale evaluation.)

Appendix

.1. Pose Estimation

We investigate the effect of backbone on pose estimation task. The baseline model is SimplePose [61] with ResNet50 and ResNet101 implemented in GluonCV [19]. As comparison we replace the backbone with ResNeSt50 and ResNeSt101 respectively while keeping other settings unchanged. The input image size is fixed to 256x192 for all runs. We use Adam optimizer with batch size 32 and initial learning rate 0.001 with no weight decay. The learning rate is divided by 10 at the 90th and 120th epoch. The experiments are conducted on COCO Keypoints dataset, and we report the OKS AP for results without and with flip test. Flip test first makes prediction on both original and horizontally flipped images, and then averages the predicted keypoint coordinates as the final output.

From Table 7, we see that models backboned with ResNeSt50/ResNeSt101 significantly outperform their ResNet counterparts. Besides, with ResNeSt50 backbone the model achieves performance similar with ResNet101 backbone.

.2. Object Detection and Instance Segmentation

For object detection, we add deformable convolution to our Cascade-RCNN model with ResNeSt-101 backbone and train the model on the MS-COCO training set for 1x schedule. The resulting model achieves 49.2% mAP on COCO test-dev set, which surpass all previous methods including these employing multi-scale evaluation. Detailed results are shown in Table 9.

	Method	Backbone	Deformable	box mAP%	mask mAP%
Prior Work	DCNv2 [76]	ResNet101 [23]	v2	45.8	39.7
		ResNeXt101 [63]	v2	46.7	40.5
	SNIPER [49]	ResNet101* [13]	v1	47.1	41.3
	Cascade-RCNN [5]	ResNext101 [63]	n/a	45.8	38.6
	Cascade-RCNN [5]	ResNeSt101 (ours)	v2	**50.0**	**43.0**

Table 9: Instance Segmentation results on the MS-COCO test-dev set. * denote multi-scale inference.

Method	lr schedule	SyncBN	Backbone	box mAP%	mask mAP%
Mask-RCNN [22]	1×		ResNet50 [60]	38.60	35.20
			ResNet101 [60]	40.79	36.93
			ResNeSt50 (ours)	40.85	36.99
			ResNeSt101 (ours)	**43.98**	**39.33**
		✓	ResNet50 [60]	39.97	36.05
			ResNet101 [60]	41.78	37.51
			ResNeSt50 (ours)	42.81	38.14
			ResNeSt101 (ours)	**45.75**	**40.65**
	3×		ResNet50 [60]	41.00	37.20
			ResNet101 [60]	42.90	38.60
			ResNeSt50 (ours)	43.32	38.91
			ResNeSt101 (ours)	**45.37**	**40.56**
Cascade-RCNN [4]	1×		ResNet50 [60]	42.10	36.40
			ResNet101 [60]	44.00	38.08
			ResNeSt50 (ours)	44.56	38.27
			ResNeSt101 (ours)	**46.86**	**40.23**
		✓	ResNet50 [60]	43.06	37.19
			ResNet101 [60]	44.79	38.52
			ResNeSt50 (ours)	46.19	39.55
			ResNeSt101 (ours)	**48.30**	**41.56**
	3×		ResNet50 [60]	44.3	38.5
			ResNet101 [60]	45.57	39.54
			ResNeSt50 (ours)	46.39	39.99
			ResNeSt101 (ours)	**47.70**	**41.16**

Table 10: Instance Segmentation results on the MS-COCO validation set. Comparing models trained w/ and w/o SyncBN, and using 1× and 3× learning rate schedules.

Method	Deformable [76] (v2)	box mAP%	mask mAP%
Cascade-RCNN [4]		48.30	41.56
	✓	**49.39**	**42.56**

Table 11: The results of Cascade-Mask-RCNN on COCO val set. The ResNeSt-101 is applied with and without deformable convolution v2 [76]. It shows that our split-attention module is compatible with other existing modules.

We include more results of instance segmentation, shown in Table 10, from the models trained with 1x/3x learning rate schedules and with/without SyncBN. All of resutls are reported on COCO val dataset. For both 50/101-layer settings, our ResNeSt backbones still outperform the corresponding baselines with different lr schedules. Same as the Table. 6 in the main text, our ResNeSt50 also exceeds the result of the standard ResNet101.

We also evaluate our ResNeSt with and without deformable convolution v2 [76]. With its help, we are able to obtain a higher performance, shown in Table 11. It indicates our designed module is compatible with deformable convolution.