==> 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.