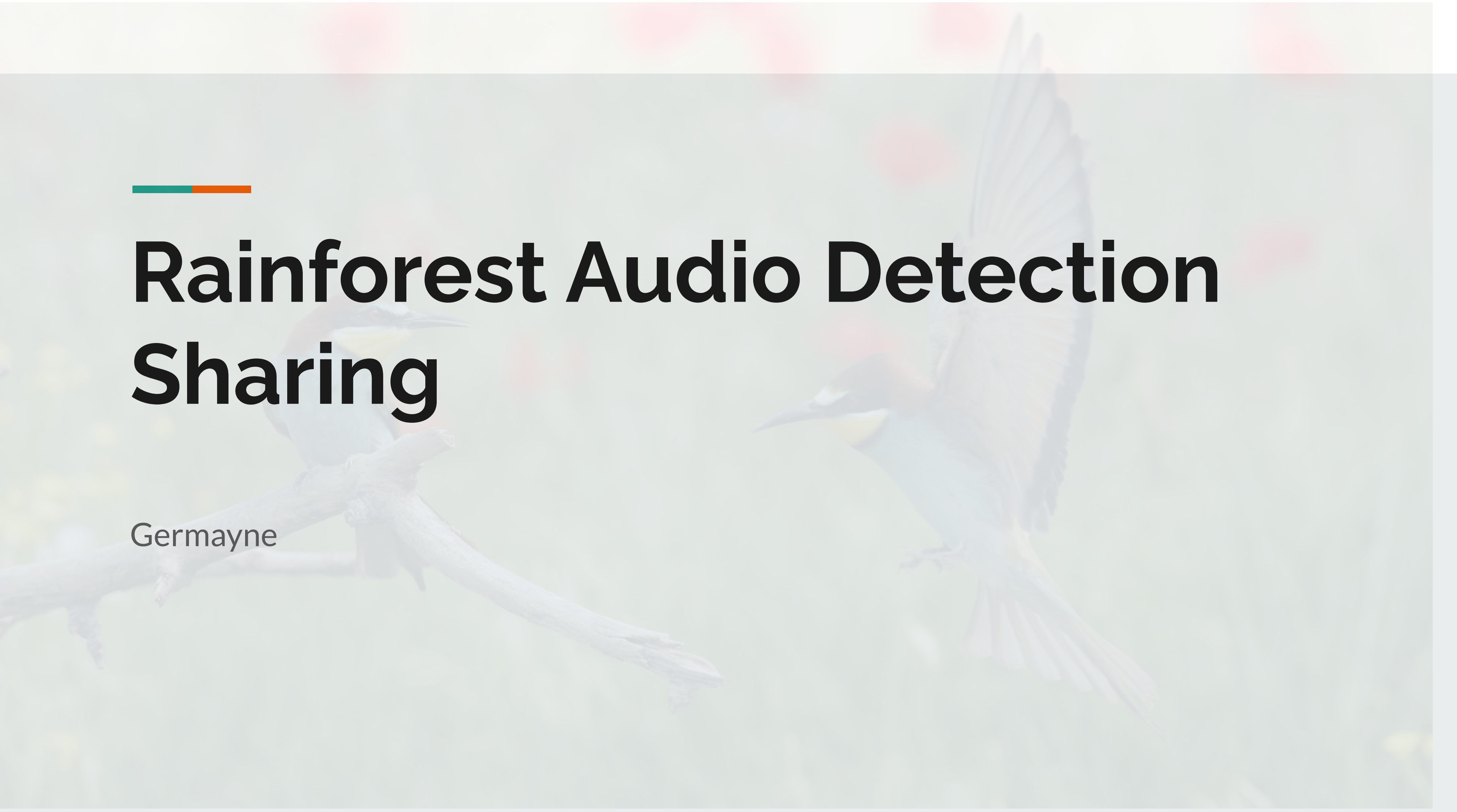


Rainforest Audio Detection Sharing

Germayne

Agenda

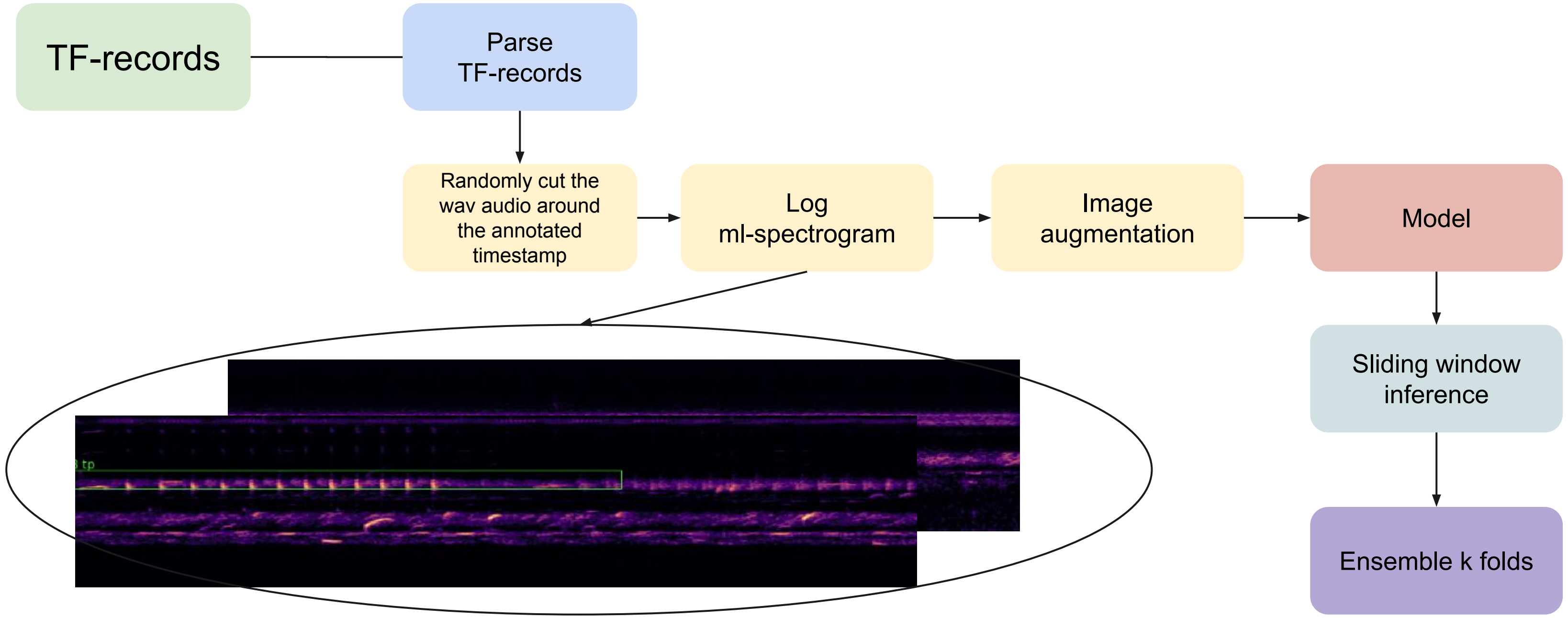
Competition Background

- Given wav audio files from different species (birds and amphibian species)
- Training data with true positive labels of the species in the audio clip
- Training data with false positive labels of the species in the audio clip
    - Time-stamp start and end of annotated species label (t_min, t_max)
    - Frequency of the annotated signal (f_min, f_max)

Competition Background

- Evaluation: label weighted - label-ranking average precision
    - If each sample contains only 1 relevant label, becomes MRR
- To have an intuition of the metric:
    - Do this separately for each species class , for each sample
        - (# of true labels as we move down the rank) / (as we move down the rank, how many probabilities we took to reach the true label)
    - We average the “score” of each species by the number of sample it appeared in
    - Finally we combine all species by a weighted average of the species appearance out of the total number of samples
- 1 - best , 0 - worst

General Pipeline - Preprocessing



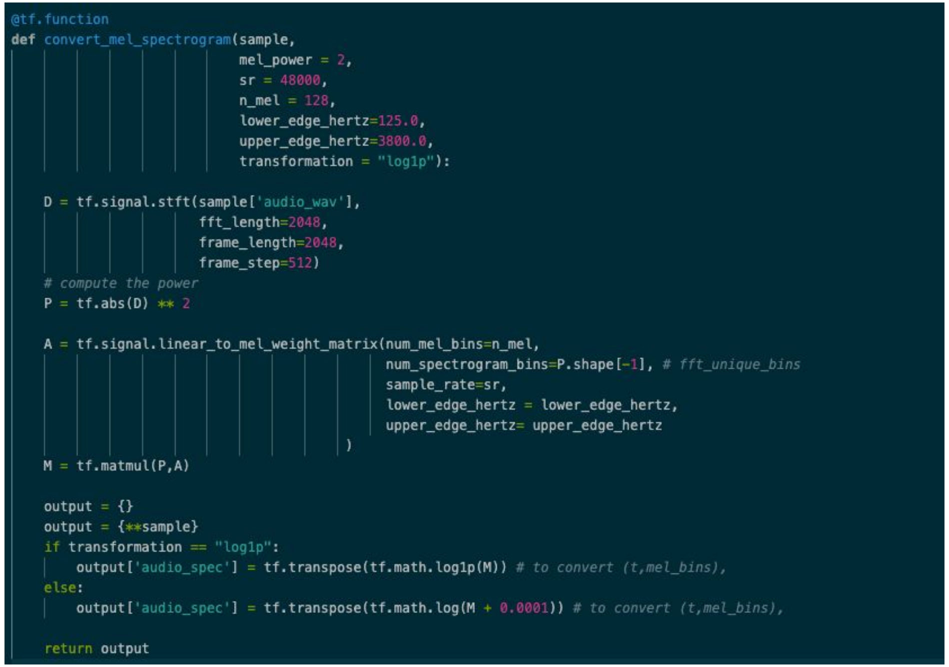
Log Mel-Spectrogram

- Sliding window STFT to convert to frequency domain
    - Returns 2D data per wav audio
    - (t step, FFT unique bins)
    - Compute the power / magnitude by squaring the previous output
    - Generate mel scale - a non linear transformation of the frequency scale
    - Finally just a dot product using: Magnitude and Mel Scale
    - Take log1p on the mel spectrogram

Good reference:
https://carlthome.github.io/posts/signal-reconstruction-from-spectrograms%20/

https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0

Log Mel-Spectrogram



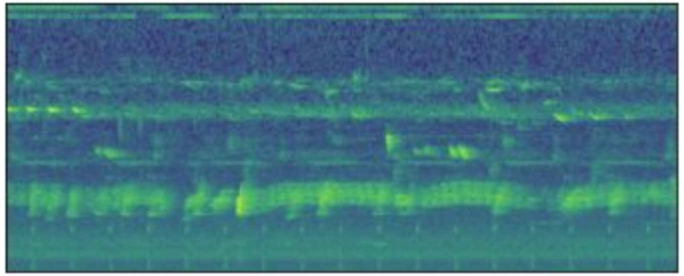
```python
@tf.function
def convert_mel_spectrogram(sample,
                            mel_power = 2,
                            sr = 48000,
                            n_mel = 128,
                            lower_edge_hertz=125.0,
                            upper_edge_hertz=3800.0,
                            transformation = "log1p"):

    D = tf.signal.stft(sample['audio_wav'],
                       fft_length=2048,
                       frame_length=2048,
                       frame_step=512)
    # compute the power
    P = tf.abs(D) ** 2

    A = tf.signal.linear_to_mel_weight_matrix(num_mel_bins=n_mel,
                                              num_spectrogram_bins=P.shape[-1], # fft_unique_bins
                                              sample_rate=sr,
                                              lower_edge_hertz = lower_edge_hertz,
                                              upper_edge_hertz= upper_edge_hertz
                                             )
    M = tf.matmul(P,A)

    output = {}
    output = {**sample}
    if transformation == "log1p":
        output['audio_spec'] = tf.transpose(tf.math.log1p(M)) # to convert (t,mel_bins),
    else:
        output['audio_spec'] = tf.transpose(tf.math.log(M + 0.0001)) # to convert (t,mel_bins),

    return output
```

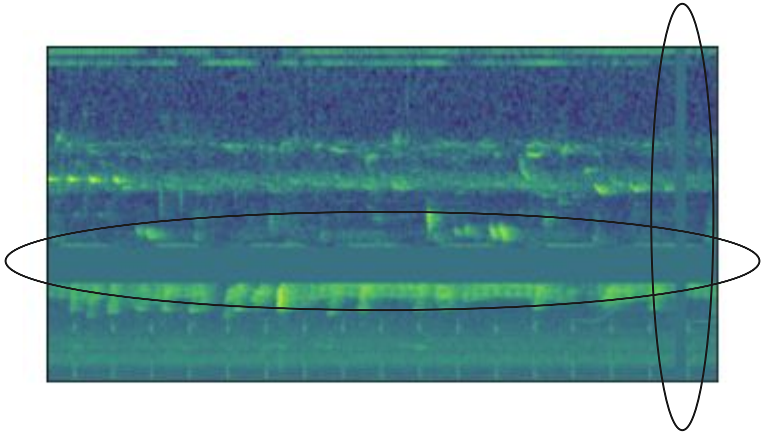
Data-Augmentation

- Spec-augmentation (using tfio)
- Random-resize crop
- Gaussian noise
- Random brightness
- Random flip up, down, left , right
- Everything with a probability of randomness
- Mix-up
    - Blend images , blend targets

Specaugmentation

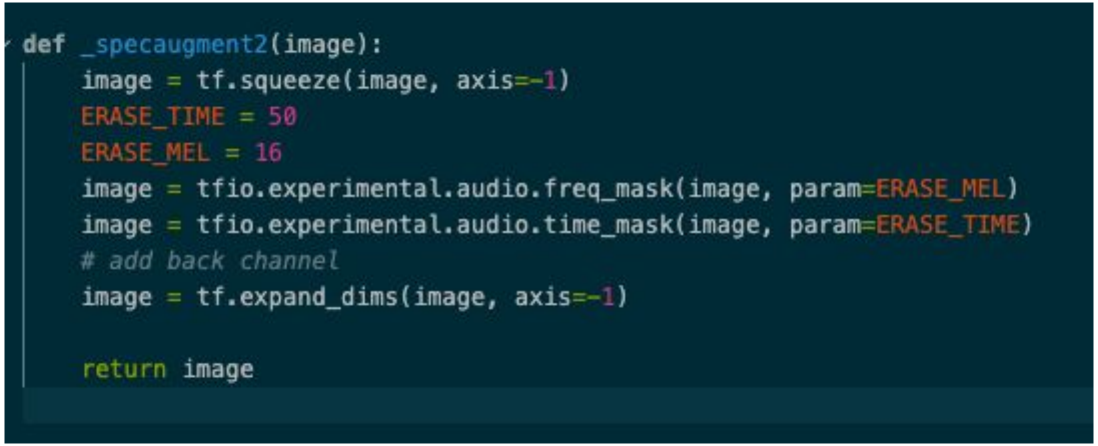
- Basically randomly mask portion of either time axis or frequency axis





Specaugmentation

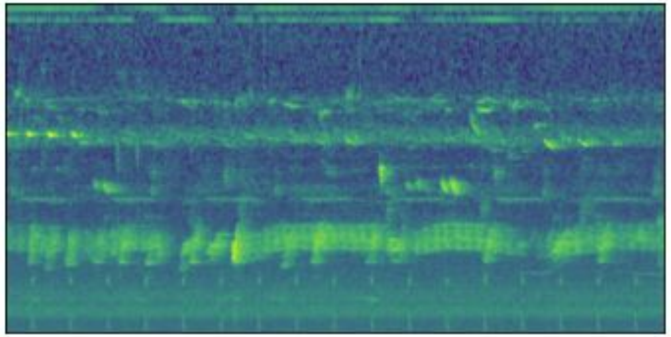
- Basically randomly mask portion of either time axis or frequency axis

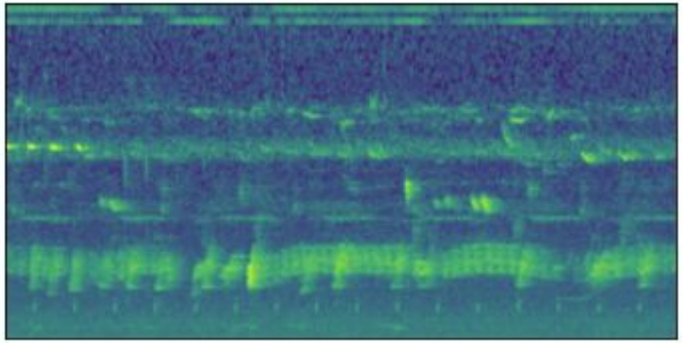


```
def _specaugment2(image):
    image = tf.squeeze(image, axis=-1)
    ERASE_TIME = 50
    ERASE_MEL = 16
    image = tfio.experimental.audio.freq_mask(image, param=ERASE_MEL)
    image = tfio.experimental.audio.time_mask(image, param=ERASE_TIME)
    # add back channel
    image = tf.expand_dims(image, axis=-1)

    return image
```

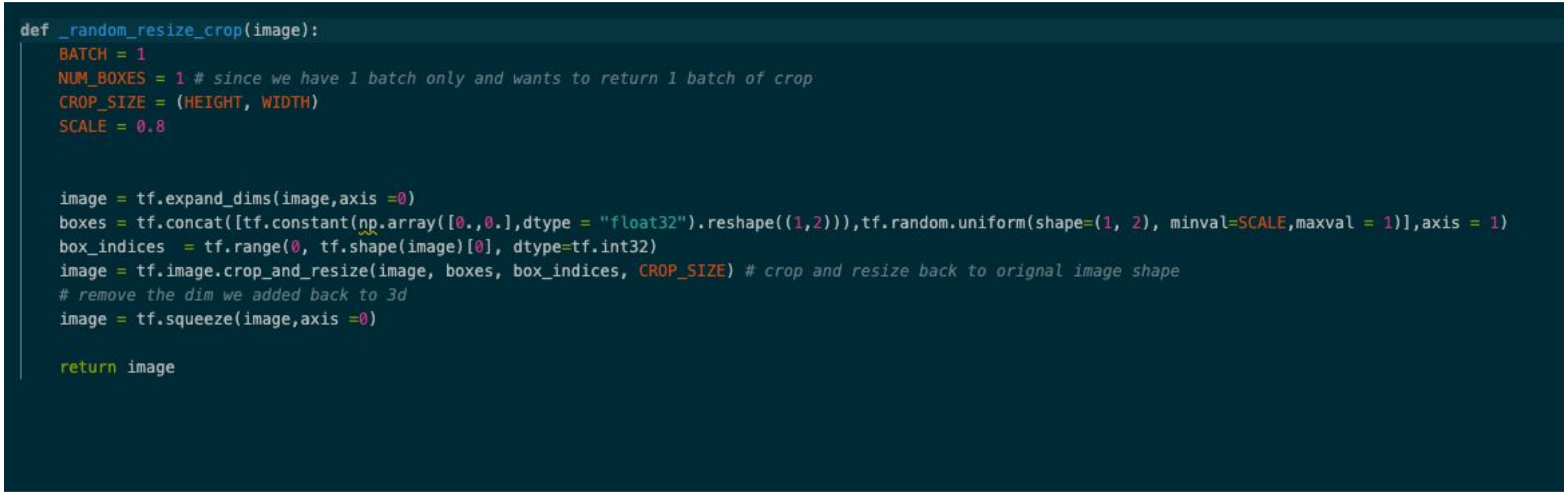
Random resize and crop

- Found out about this augmentation from a previous competition's solution
- Tensorflow's crop_and_resize
- Set the range of to be between 0.8 and 1
- Worked extremely well





Random resize and crop



```python
def _random_resize_crop(image):
    BATCH = 1
    NUM_BOXES = 1 # since we have 1 batch only and wants to return 1 batch of crop
    CROP_SIZE = (HEIGHT, WIDTH)
    SCALE = 0.8


    image = tf.expand_dims(image,axis =0)
    boxes = tf.concat([tf.constant(np.array([0.,0.],dtype = "float32").reshape((1,2))),tf.random.uniform(shape=(1, 2), minval=SCALE,maxval = 1)],axis = 1)
    box_indices  = tf.range(0, tf.shape(image)[0], dtype=tf.int32)
    image = tf.image.crop_and_resize(image, boxes, box_indices, CROP_SIZE) # crop and resize back to orignal image shape
    # remove the dim we added back to 3d
    image = tf.squeeze(image,axis =0)

    return image
```

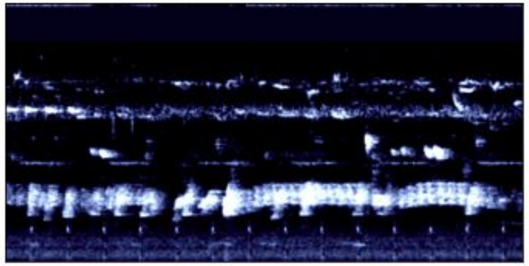
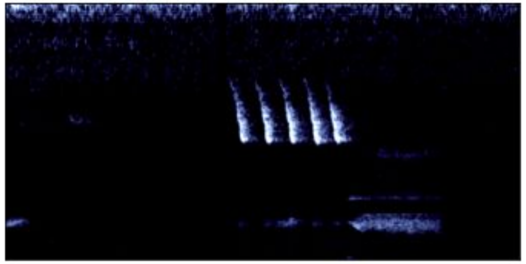
Mix up

- Convex combination of images and their corresponding labels
- Why it works?
    - Prevents memorization by introducing sampling from vicinal distribution
    - Data augmentation that encourages the model to behave linearly between training examples
    - Reduce oscillations when predicting outside training examples
- Applied within each mini-batches at the spectrogram level
- Source: https://arxiv.org/pdf/1710.09412.pdf

Mix up

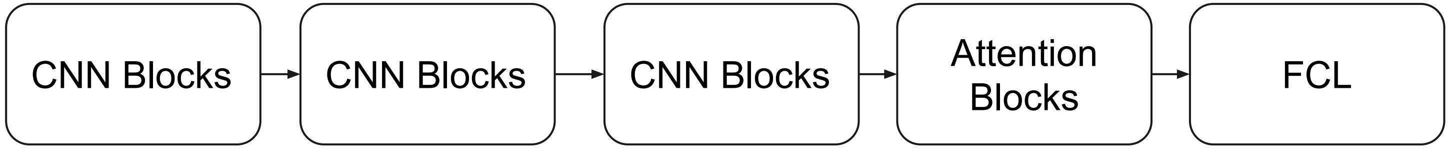
where $\lambda \sim \text{Beta}(\alpha, \alpha)$, for $\alpha \in (0, \infty)$. In a nutshell, sampling from the *mixup* vicinal distribution produces virtual feature-target vectors

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j,$$
$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j,$$

Model - Attention model v1

- Decide to implement an architecture following one of a model from a 1st place solution of a previous audio tagging competition
- Attention based CNN model
- Port the attention model to tensorflow

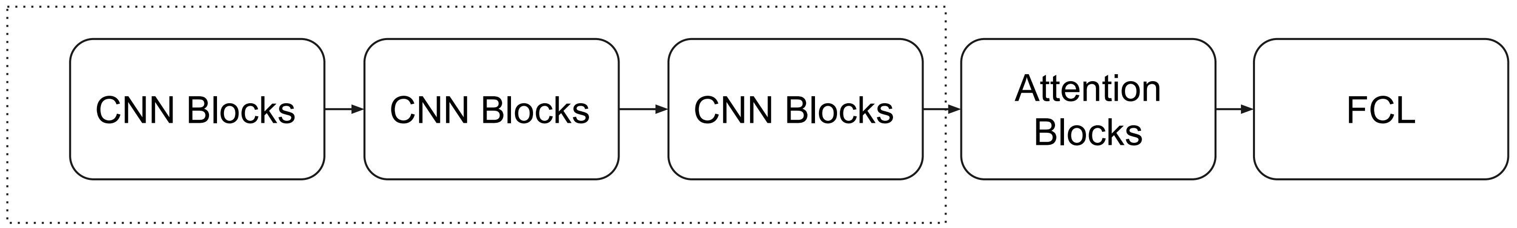


conv2D
batchnorm2D
Relu
(x2 of the above)

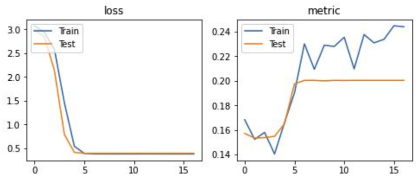
Drop-out (0.4)
Dense (prelu)
batchnorm2D
Drop-out (0.4)
Dense linear

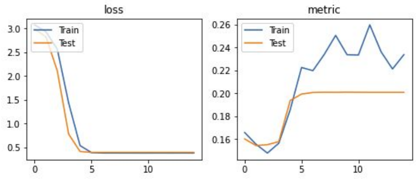
Model - Attention model v1

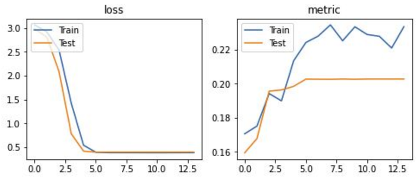
- Why have CNN blocks when we can use transfer-learning?
- Baseline starts with Resnet50
- But later it turns out that densenet is really good
- Source: https://arxiv.org/abs/2007.11154

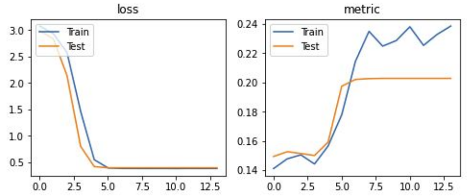


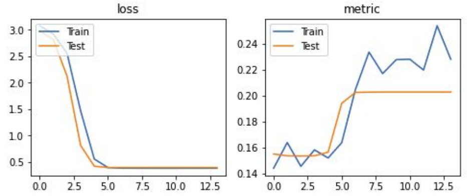
Model - Attention model v1





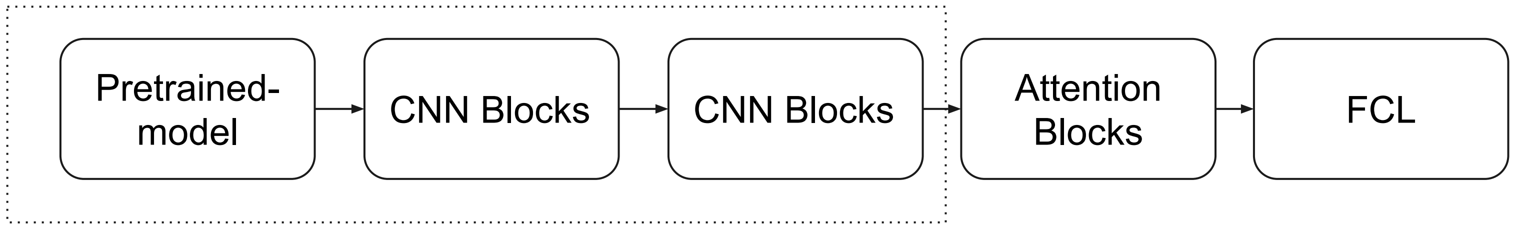




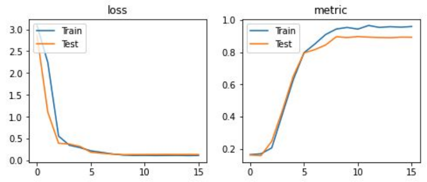


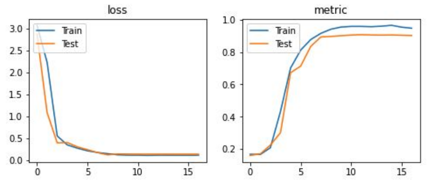
Model - Attention model v2

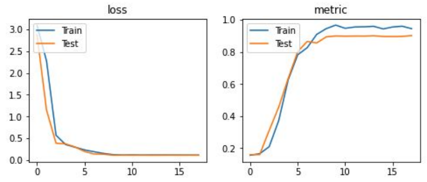
- After getting stuck at for while, it turns out
- Adding additional convo-blocks lead to a much better result...

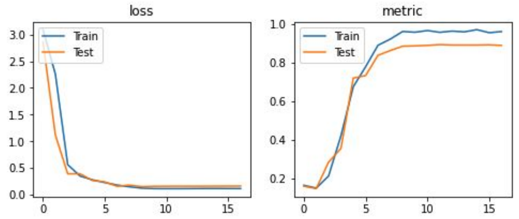


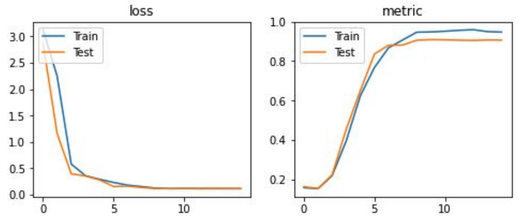
Model - Attention model v2



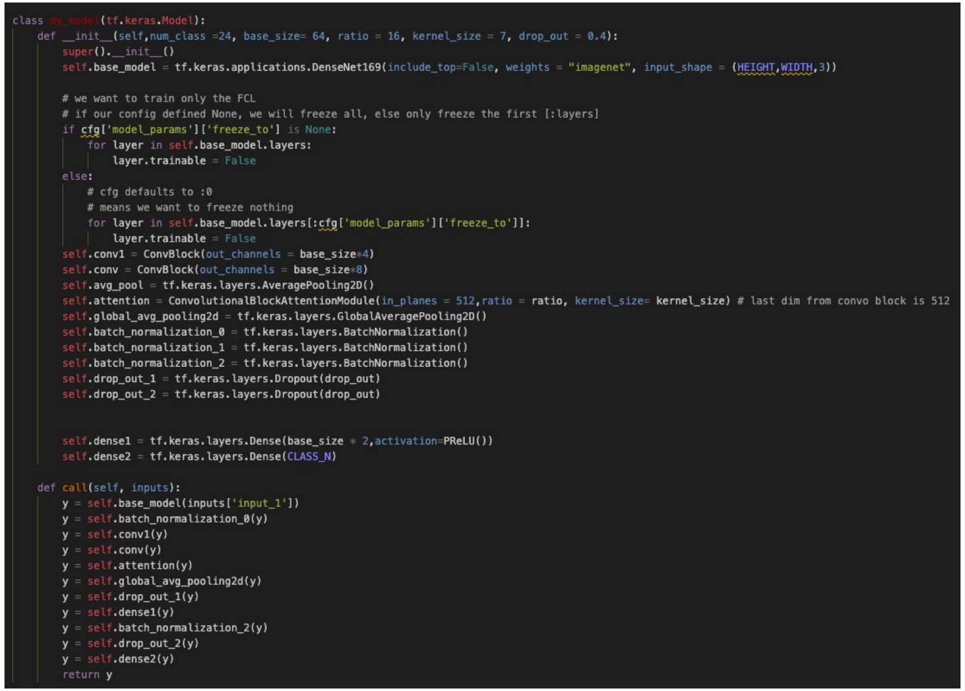








CNN - Attention Model v2



```python
class my_model(tf.keras.Model):
    def __init__(self,num_class =24, base_size= 64, ratio = 16, kernel_size = 7, drop_out = 0.4):
        super().__init__()
        self.base_model = tf.keras.applications.DenseNet169(include_top=False, weights = "imagenet", input_shape = (HEIGHT,WIDTH,3))

        # we want to train only the FCL
        # if our config defined None, we will freeze all, else only freeze the first [:layers]
        if cfg['model_params']['freeze_to'] is None:
            for layer in self.base_model.layers:
                layer.trainable = False
        else:
            # cfg defaults to :0
            # means we want to freeze nothing
            for layer in self.base_model.layers[:cfg['model_params']['freeze_to']]:
                layer.trainable = False
        self.conv1 = ConvBlock(out_channels = base_size*4)
        self.conv = ConvBlock(out_channels = base_size*8)
        self.avg_pool = tf.keras.layers.AveragePooling2D()
        self.attention = ConvolutionalBlockAttentionModule(in_planes = 512,ratio = ratio, kernel_size= kernel_size) # last dim from convo block is 512
        self.global_avg_pooling2d = tf.keras.layers.GlobalAveragePooling2D()
        self.batch_normalization_0 = tf.keras.layers.BatchNormalization()
        self.batch_normalization_1 = tf.keras.layers.BatchNormalization()
        self.batch_normalization_2 = tf.keras.layers.BatchNormalization()
        self.drop_out_1 = tf.keras.layers.Dropout(drop_out)
        self.drop_out_2 = tf.keras.layers.Dropout(drop_out)


        self.dense1 = tf.keras.layers.Dense(base_size * 2,activation=PReLU())
        self.dense2 = tf.keras.layers.Dense(CLASS_N)

    def call(self, inputs):
        y = self.base_model(inputs['input_1'])
        y = self.batch_normalization_0(y)
        y = self.conv1(y)
        y = self.conv(y)
        y = self.attention(y)
        y = self.global_avg_pooling2d(y)
        y = self.drop_out_1(y)
        y = self.dense1(y)
        y = self.batch_normalization_2(y)
        y = self.drop_out_2(y)
        y = self.dense2(y)
        return y
```

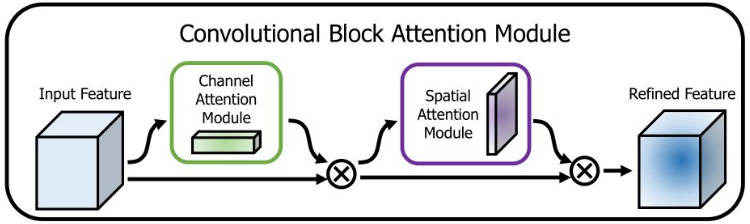
Convolutional block attention module

- Introduced in CBAM paper
- Pytorch implementation github
- A CBAM block contains 2 sub-blocks
  - Channel attention module
  - Spatial attention module
- Idea is to infer feature maps in 2 different dimension
  - Channel dimension
  - Spatial dimension
- Result: adaptive feature refinement

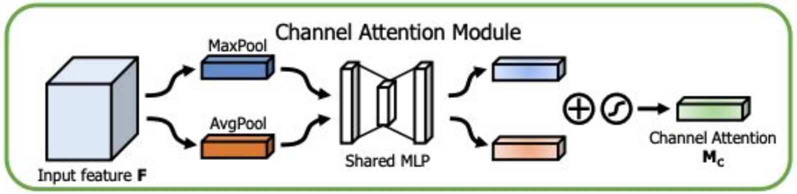
input $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$

$$\mathbf{F}' = \mathbf{M_c}(\mathbf{F}) \otimes \mathbf{F},$$
$$\mathbf{F}'' = \mathbf{M_s}(\mathbf{F}') \otimes \mathbf{F}',$$

$\otimes$ denotes element-wise multiplication



Channel Attention Module



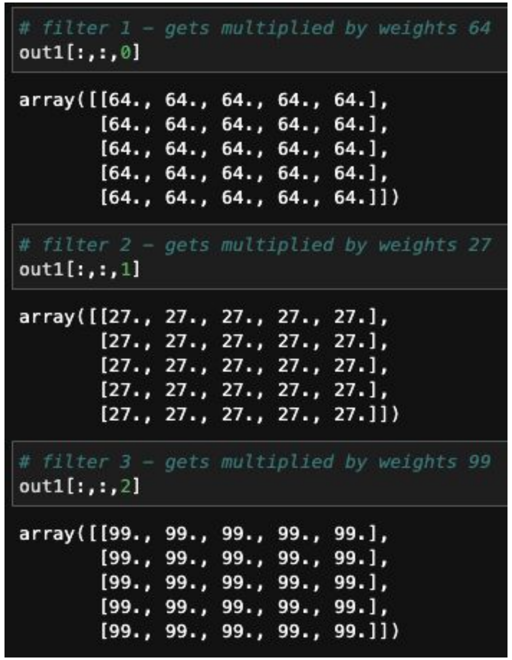
- The channel attention map contains inter-channel relationship
- **Feature map’s channel are feature detectors**
- CA module focus on what is meaningful given an input image
- Author’s idea:
  - To compute channel attention, spatial dimension is squeezed (aggregated) via max pool and avg pool
  - Max-pool gathers important distinctive object features
  - Empirically shows that both pooling give better results
- Shared MLP consists of 2 CNN that basically returns the same filter size as the input channel

$$\begin{aligned}\mathbf{M_c}(\mathbf{F}) &= \sigma(MLP(AvgPool(\mathbf{F})) + MLP(MaxPool(\mathbf{F})))\\ &= \sigma(\mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{avg}})) + \mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{max}}))),\end{aligned}$$

Channel Attention Module

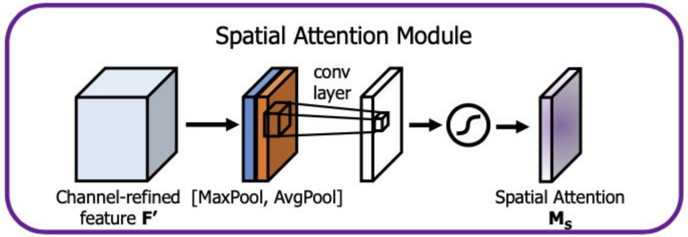
- Multipliers are applied on each filter
- The important filters gets more “attention”

$$\mathbf{M_c}(\mathbf{F}) = \sigma(MLP(AvgPool(\mathbf{F})) + MLP(MaxPool(\mathbf{F}))) \\ = \sigma(\mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{avg}})) + \mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{max}}))),$$



```
# filter 1 - gets multiplied by weights 64
out1[:,:,0]

array([[64., 64., 64., 64., 64.],
       [64., 64., 64., 64., 64.],
       [64., 64., 64., 64., 64.],
       [64., 64., 64., 64., 64.],
       [64., 64., 64., 64., 64.]])

# filter 2 - gets multiplied by weights 27
out1[:,:,1]

array([[27., 27., 27., 27., 27.],
       [27., 27., 27., 27., 27.],
       [27., 27., 27., 27., 27.],
       [27., 27., 27., 27., 27.],
       [27., 27., 27., 27., 27.]])

# filter 3 - gets multiplied by weights 99
out1[:,:,2]

array([[99., 99., 99., 99., 99.],
       [99., 99., 99., 99., 99.],
       [99., 99., 99., 99., 99.],
       [99., 99., 99., 99., 99.],
       [99., 99., 99., 99., 99.]])
```

Spatial Attention Module



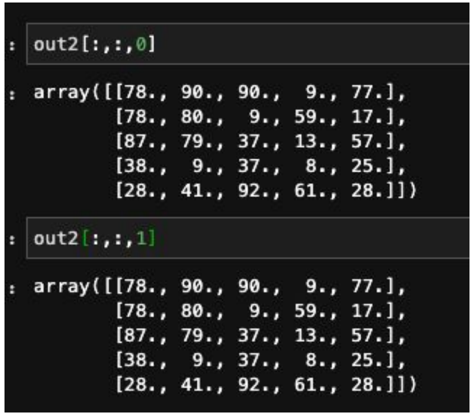
- The spatial attention map search for the spatial location of the informative rather than the what (covered by CA)
- Author’s idea
  - Average and max pooling are down on the channel axis (basically squeeze the channel)
  - Apply convolutional operation on the feature descriptor

$$\begin{aligned}\mathbf{M_s}(\mathbf{F}) &= \sigma(f^{7\times7}([AvgPool(\mathbf{F}); MaxPool(\mathbf{F})])) \\ &= \sigma(f^{7\times7}([\mathbf{F^s_{avg}}; \mathbf{F^s_{max}}])),\end{aligned} \tag{3}$$

where σ denotes the sigmoid function and $f^{7\times7}$ represents a convolution operation with the filter size of 7×7.

Spatial Attention Module

- Multipliers are applied on pixel level
- The important pixels gets more "attention"



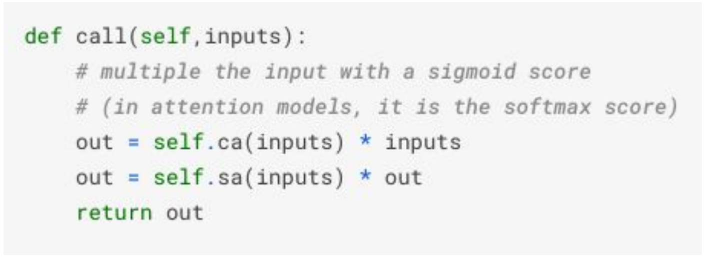
```
: out2[:,:,0]

: array([[78., 90., 90.,  9., 77.],
         [78., 80.,  9., 59., 17.],
         [87., 79., 37., 13., 57.],
         [38.,  9., 37.,  8., 25.],
         [28., 41., 92., 61., 28.]])

: out2[:,:,1]

: array([[78., 90., 90.,  9., 77.],
         [78., 80.,  9., 59., 17.],
         [87., 79., 37., 13., 57.],
         [38.,  9., 37.,  8., 25.],
         [28., 41., 92., 61., 28.]])
```

$$\begin{aligned}\mathbf{M_s}(\mathbf{F}) &= \sigma(f^{7\times7}([AvgPool(\mathbf{F}); MaxPool(\mathbf{F})])) \\ &= \sigma(f^{7\times7}([\mathbf{F^s_{avg}}; \mathbf{F^s_{max}}])),\end{aligned} \qquad (3)$$

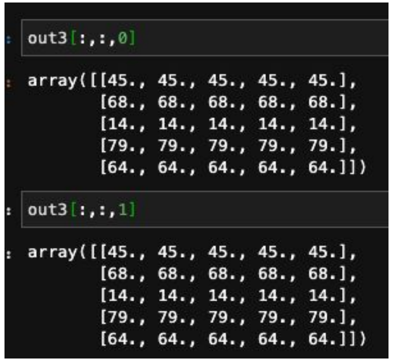
where σ denotes the sigmoid function and $f^{7\times7}$ represents a convolution operation with the filter size of 7×7.

Tweaks

- Spatial attention - Instead of squeezing the channel layer, i squeezed the time axis
  - The resulting attention will be along the time axis (Batch,HWC=> axis = 1)
- Channel attention - no change
- Instead of using a back to back architecture for the attention blocks, they will be placed parallel
  - We pass feature inputs F to both attention layers separately and multiply them together only at the end instead of stacking them one after another



```
def call(self,inputs):
    # multiple the input with a sigmoid score
    # (in attention models, it is the softmax score)
    out = self.ca(inputs) * inputs
    out = self.sa(inputs) * out
    return out
```



```
out3[:,:,0]

array([[45., 45., 45., 45., 45.],
       [68., 68., 68., 68., 68.],
       [14., 14., 14., 14., 14.],
       [79., 79., 79., 79., 79.],
       [64., 64., 64., 64., 64.]])

out3[:,:,1]

array([[45., 45., 45., 45., 45.],
       [68., 68., 68., 68., 68.],
       [14., 14., 14., 14., 14.],
       [79., 79., 79., 79., 79.],
       [64., 64., 64., 64., 64.]])
```

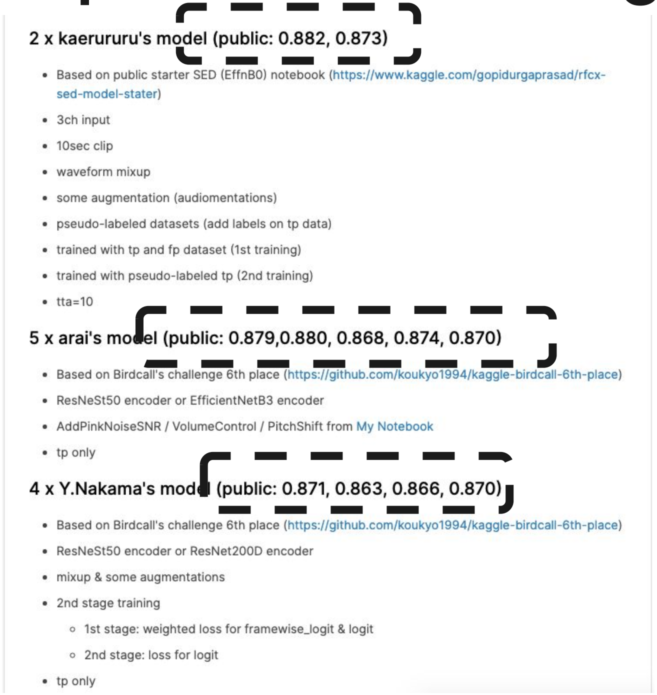
Tweaks

Single Model	Public LB	Private LB (released after)
Modified spatial blocks	0.84994	0.85780
Modified spatial blocks + modified parallel blocks	0.88324	0.89100

Submission

- Simple weighted average of my 7 dense-net models
  - With different augmentation
  - Different wav-length range extraction (extracted 6,7 seconds out of 60 seconds around the annotated regions)
  - Different dense-net models (121,169,201)
  - Nearing the end, lower batch size and higher iteration improved the models as well
  - Single best densenet model was around **<u>0.891 LWRAP (private)</u>**
  - Final result was around 0.89976 (private)
  - Had a submission of 0.90101 (private) but doesn’t change the placement much
- Tough battle to maintain position, especially the last few days

Comparison to some higher rank solutions



2 x kaerururu's model (public: 0.882, 0.873)

- Based on public starter SED (EffnB0) notebook (https://www.kaggle.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 Notebook
- tp only

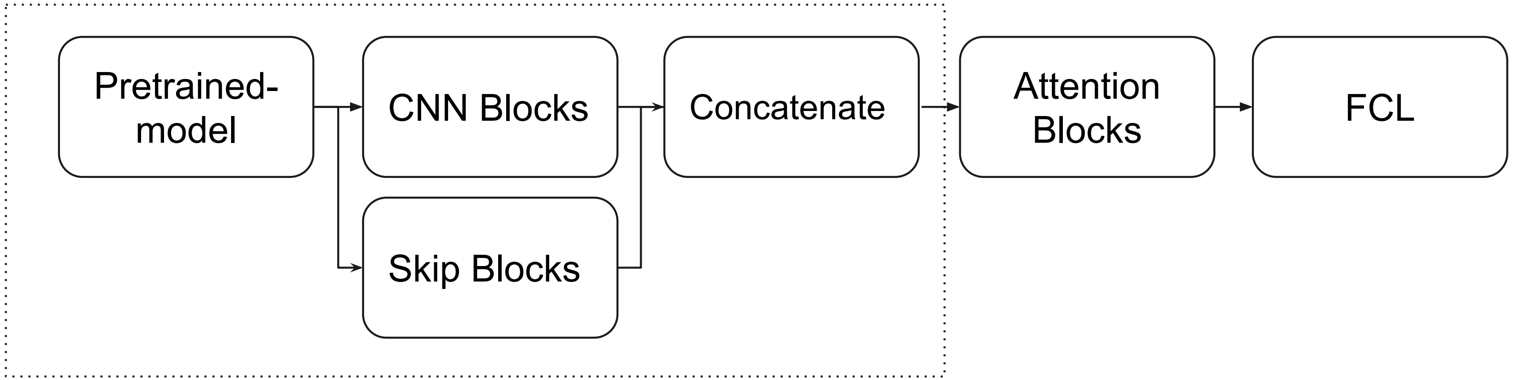
4 x Y.Nakama's model (public: 0.871, 0.863, 0.866, 0.870)

- 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

- Ranked 16th
- Diverse models
  - Ensemble: 0.946

Other things that did not work

- Tried to implement a skip-attention model
  - Extension to the attention model architecture
  - Added “skip” convo-blocks
  - Not as good as the attention architecture



Learning points: Dealing with noisy labels

- For a given clip, we have the TP
- For a given clip, we have the FP
- This means the rest of the classes are unknown
  - But they could be present in the audio
- Hence, our labels are noisy and we should have low confidence
- Top winners- Lsoft loss (noise-robust loss). Paper
  - Modification of cross entropy loss
  - Dynamically updates the target labels based on model learning progress
  - Convex combination of prediction and potentially noisy labels
  - Idea: pay less attention to noisy labels, more attention to prediction
  - This is known as soft bootstrapping

$$L_{soft} = -\sum_{k=1}^{K}[\beta y_k + (1-\beta)\hat{y}_k]\log(\hat{y}_k), \quad \beta \in [0,1].$$

Learning points: Dealing with noisy labels

- Various way of implementing this L soft loss
    a. For a given sample, those with labels uses actual target. (TP/FP?) Those without labels we use L soft loss instead of the actual labels (0)
    b. L soft only on TP data (like how i did)

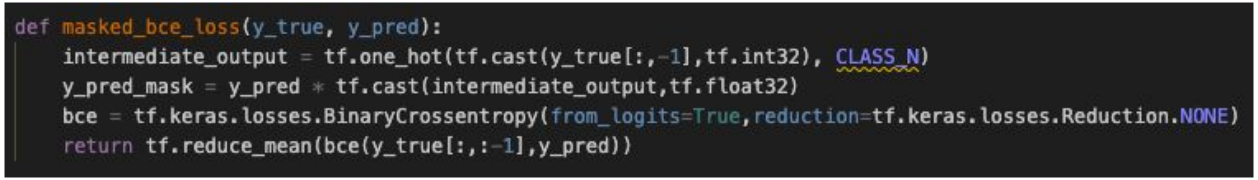
Learning points: Loss Masking

- Another way to hide noisy labels - loss masking
- Utilizing the FP was the key for this competition
    - This requires loss masking
    - Source: https://www.sciencedirect.com/science/article/abs/pii/S0003682X20304795
    - Only penalize for labels that are known, let TP be 1 and FP be 0
    - Let the rest be unknown (mistake for me was to let the other labels be 0) - hence masking
    - BCE loss works well for this
    - Proxy using ROC-AUC
        - Essentially, each sample becomes mono-labels

$$Loss = \begin{cases} -(y\log(p) + (1-y)\log(1-p)), & y \in \{0, 1\} \\ 0, & y = \text{Unknown} \end{cases}$$

Learning points: Loss Masking

- Could not get it to work. Might be something i am doing wrong or requires certain augmentation to make it work

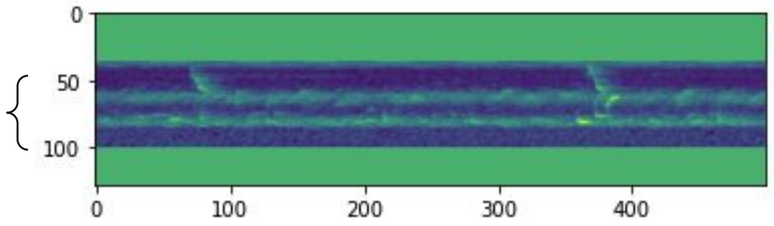


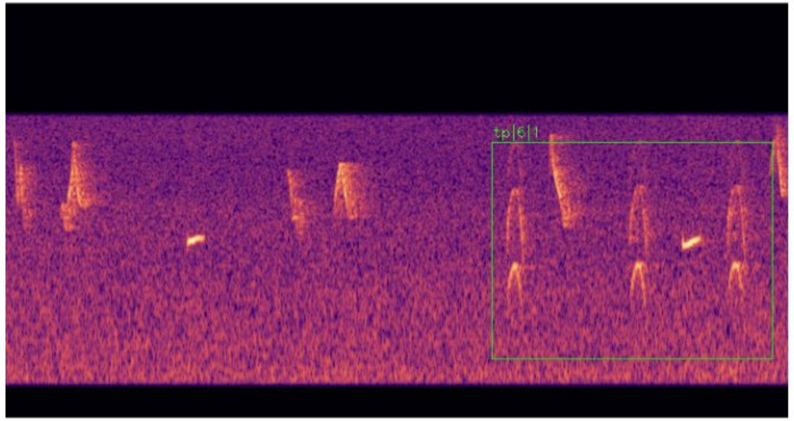
```python
def masked_bce_loss(y_true, y_pred):
    intermediate_output = tf.one_hot(tf.cast(y_true[:,-1],tf.int32), CLASS_N)
    y_pred_mask = y_pred * tf.cast(intermediate_output,tf.float32)
    bce = tf.keras.losses.BinaryCrossentropy(from_logits=True,reduction=tf.keras.losses.Reduction.NONE)
    return tf.reduce_mean(bce(y_true[:,:-1],y_pred))
```

Learning points: Frequency filtering

- Crop and mask regions of spectrogram that are relevant to the annotated labels
    - Because each species are restricted to a certain frequency range (y-axis)
    - In inference time, we need to slide across time (x-axis) and take max of predictions

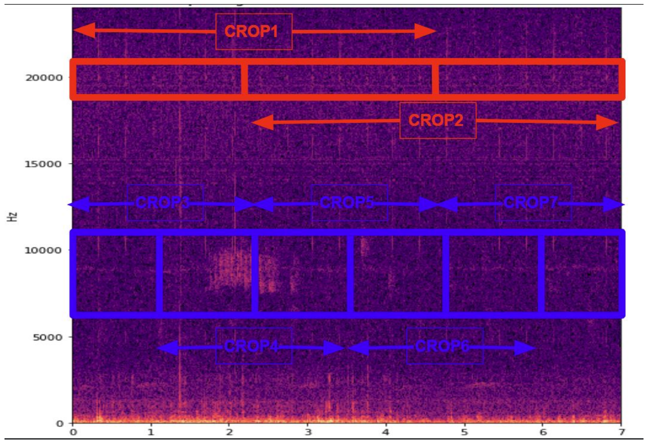
FP range





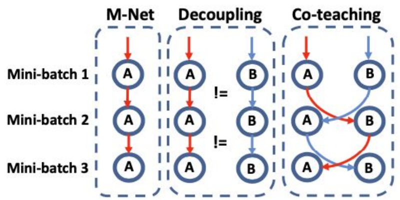
Learning points: Frequency filtering

- In inference time, we need to slide across time (x-axis) and take max of predictions
    - This is because each species corresponds to a fixed region of frequency

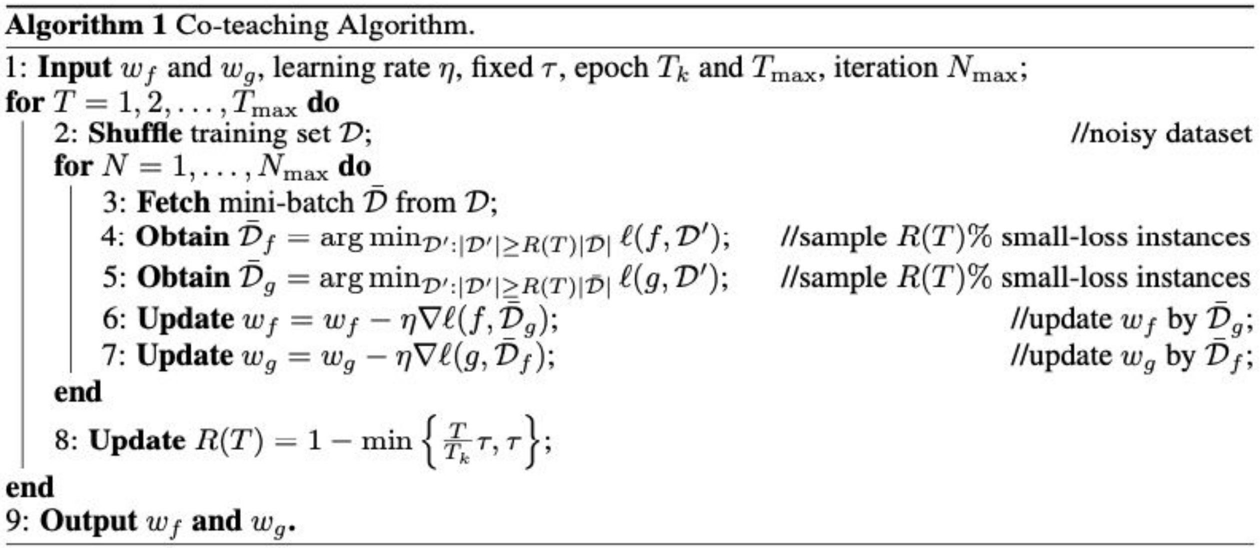


Learning points: Co-teaching model Algorithm

- Paper
- Github
- Credit
- Idea: train robust models with noisy dataset (45%)
- Train 2 neural network that will co-train the peer
- For **a given mini batch**, **each model selects** some R(T) % of samples (with small losses) for its peer
- Each model will update their parameter using instances selected by its peer
- Each model determines what is a good instance (clean) and their peer will select these instances to compute forward and backward propagation
- Because peers teach each other useful instances - it is co-teaching



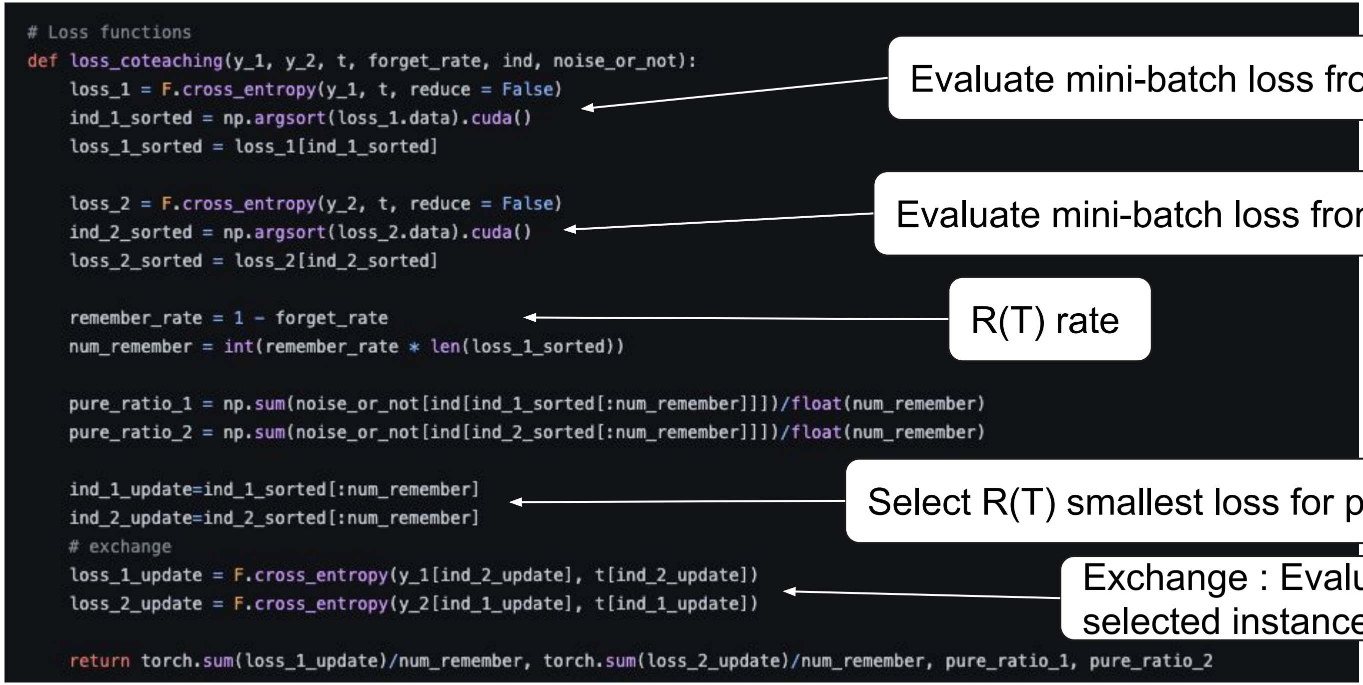
Learning points: Co-teaching model Algorithm



Algorithm 1 Co-teaching Algorithm.

```
1: Input $w_f$ and $w_g$, learning rate $\eta$, fixed $\tau$, epoch $T_k$ and $T_{\max}$, iteration $N_{\max}$;
for $T = 1, 2, \ldots, T_{\max}$ do
    2: Shuffle training set $\mathcal{D}$;                                          //noisy dataset
    for $N = 1, \ldots, N_{\max}$ do
        3: Fetch mini-batch $\bar{\mathcal{D}}$ from $\mathcal{D}$;
        4: Obtain $\bar{\mathcal{D}}_f = \arg\min_{\mathcal{D}':|\mathcal{D}'|\geq R(T)|\bar{\mathcal{D}}|} \ell(f, \mathcal{D}')$;    //sample $R(T)\%$ small-loss instances
        5: Obtain $\bar{\mathcal{D}}_g = \arg\min_{\mathcal{D}':|\mathcal{D}'|\geq R(T)|\bar{\mathcal{D}}|} \ell(g, \mathcal{D}')$;    //sample $R(T)\%$ small-loss instances
        6: Update $w_f = w_f - \eta\nabla\ell(f, \bar{\mathcal{D}}_g)$;                //update $w_f$ by $\bar{\mathcal{D}}_g$;
        7: Update $w_g = w_g - \eta\nabla\ell(g, \bar{\mathcal{D}}_f)$;                //update $w_g$ by $\bar{\mathcal{D}}_f$;
    end
    8: Update $R(T) = 1 - \min\left\{\frac{T}{T_k}\tau, \tau\right\}$;
end
9: Output $w_f$ and $w_g$.
```

Learning points: Co-teaching model loss



```python
# Loss functions
def loss_coteaching(y_1, y_2, t, forget_rate, ind, noise_or_not):
    loss_1 = F.cross_entropy(y_1, t, reduce = False)
    ind_1_sorted = np.argsort(loss_1.data).cuda()
    loss_1_sorted = loss_1[ind_1_sorted]

    loss_2 = F.cross_entropy(y_2, t, reduce = False)
    ind_2_sorted = np.argsort(loss_2.data).cuda()
    loss_2_sorted = loss_2[ind_2_sorted]

    remember_rate = 1 - forget_rate
    num_remember = int(remember_rate * len(loss_1_sorted))

    pure_ratio_1 = np.sum(noise_or_not[ind[ind_1_sorted[:num_remember]]])/float(num_remember)
    pure_ratio_2 = np.sum(noise_or_not[ind[ind_2_sorted[:num_remember]]])/float(num_remember)

    ind_1_update=ind_1_sorted[:num_remember]
    ind_2_update=ind_2_sorted[:num_remember]
    # exchange
    loss_1_update = F.cross_entropy(y_1[ind_2_update], t[ind_2_update])
    loss_2_update = F.cross_entropy(y_2[ind_1_update], t[ind_1_update])

    return torch.sum(loss_1_update)/num_remember, torch.sum(loss_2_update)/num_remember, pure_ratio_1, pure_ratio_2
```

Evaluate mini-batch loss from model 1

Evaluate mini-batch loss from model 2

R(T) rate

Select R(T) smallest loss for peers

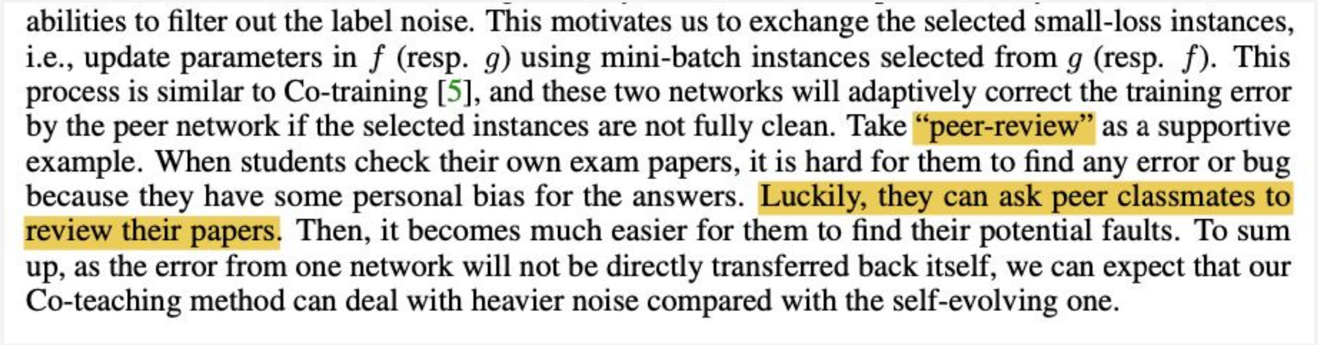
Exchange : Evaluate peer's selected instances

Learning points: Co-teaching model why it works

- Why peer selection of R(T) samples helps to find clean instance?
  - If labels are correct <-> small loss labels are likely the correct labels
  - If we use small losses for each mini-batch, model trained should be resistant to noise labels
  - Early epoch models will learn clean , easy patterns
    - But as epoch increase, model starts to overfit on noise
    - Solution: R(T) starts large, then starts getting smaller
    - This helps to keep clean instance and drop noisy ones before models memorize them

Learning points: Co-teaching model why it works

- Why 2 networks and why cross update??
  - Can use 1 network and let it evolve: similar to boosting
    - But boosting and active learning are sensitive to noise
  - Different classifier - different decision boundaries
    - Expect different performance in noise filtering


abilities to filter out the label noise. This motivates us to exchange the selected small-loss instances, i.e., update parameters in f (resp. g) using mini-batch instances selected from g (resp. f). This process is similar to Co-training [5], and these two networks will adaptively correct the training error by the peer network if the selected instances are not fully clean. Take "peer-review" as a supportive example. When students check their own exam papers, it is hard for them to find any error or bug because they have some personal bias for the answers. Luckily, they can ask peer classmates to review their papers. Then, it becomes much easier for them to find their potential faults. To sum up, as the error from one network will not be directly transferred back itself, we can expect that our Co-teaching method can deal with heavier noise compared with the self-evolving one.

Learning points: Pseudo labelling method 1

- Perform pseudo-labelling
- Source: https://www.sciencedirect.com/science/article/abs/pii/S0003682X20304795
    - Expands the loss masking technique
    - Exclude unknown labels from loss
    - Train model for a few rounds, (small epoch, paper chose 5)
    - Predict on all labels and for labels > 0.9, label as 1

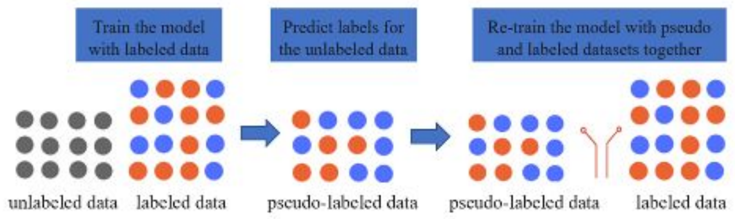

Train the model with labeled data
Predict labels for the unlabeled data
Re-train the model with pseudo and labeled datasets together
unlabeled data
labeled data
pseudo-labeled data
pseudo-labeled data
labeled data

Fig. 2. The diagram of the proposed methodology with pseudo-label generating and model training. The dots with different colors represent observations with different labels. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

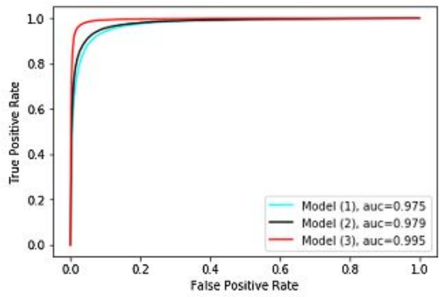


Fig. 3. The ROC Curve for each model. The proposed model (Pre-trained ResNet50 + Custom Loss Function + Pseudo Labeling) has the highest AUC value among three models.

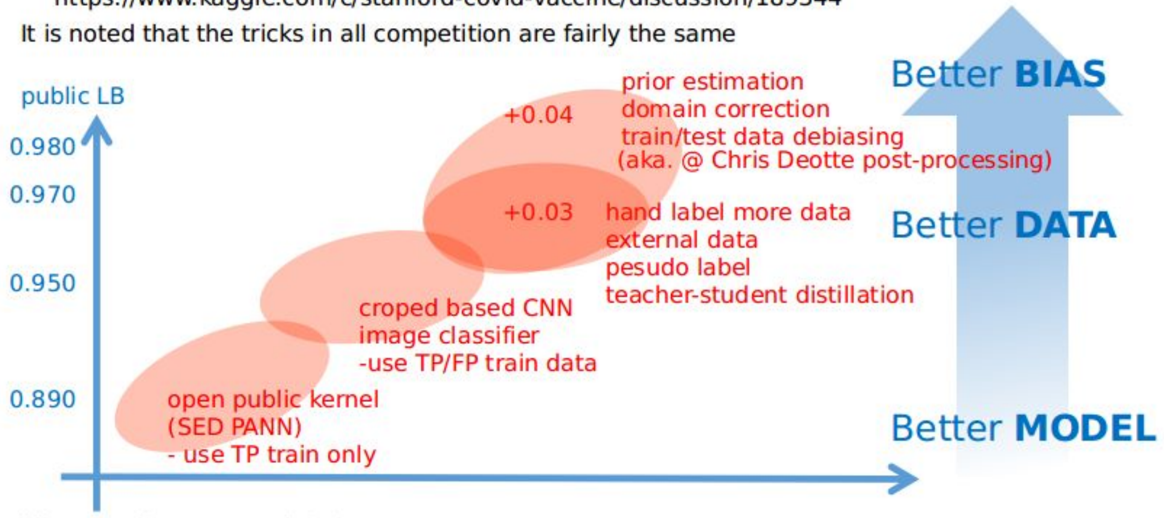
Learning points: Pseudo labelling method 2

- Train SED (sound event detection) model using TP samples
- 2 types of possible output using SED
  - Present classes and
  -

Learning points

you can compare the diagram below with the one in:
https://www.kaggle.com/c/stanford-covid-vaccine/discussion/189344

It is noted that the tricks in all competition are fairly the same



different milestones and their scores