# MIL network – evaluation and training process

## Separate Training into two stages:

First train single examples/instances so that it learns a base classification model for single instances. You can save the trained parameters. Afterwards you can load the network and add a MIL network at the end (and use a MIL data loader as well).

This way 1) you can save a lot of time because it takes forever to train a MIL network end to end. Also you can test various MIL algorithms/networks with a pretrained base model very quickly. Starting from scratch with every network would be infeasible.

So far the data also suggests, that the performance is better when training the MIL part separately (or not entirely separate but setting the base models learning rate very low in comparison)

(quick note: right now, the last layer (512 to 1 dense layer) is replaced when adding the MIL algorithm. This was added for the attention-based MIL and can at least for the prediction level mil be removed).

## Erroneous Evaluation Method

So far, for the training all samples were randomly time shifted so all the time instances are seen during training for a maximum dataset size. In evaluation however, only the first-time frame (the first e.g., 224 time samples which sometimes corresponds to only 2 seconds) are seen.

This fixed frame was chosen because this way, if the network does not change, it will always yield the same evaluation metrics. However, this way we only see how the first few seconds of a recording performs. So, we change the hyperparameters so that this smaller subset of samples performs well while ignoring all the other data.

## Fixing the issue:

The goal is, that we get evaluation metrics for any time instances of the evaluation samples while also trying to have a consistent result when evaluating the same model (with same weights) multiple times.

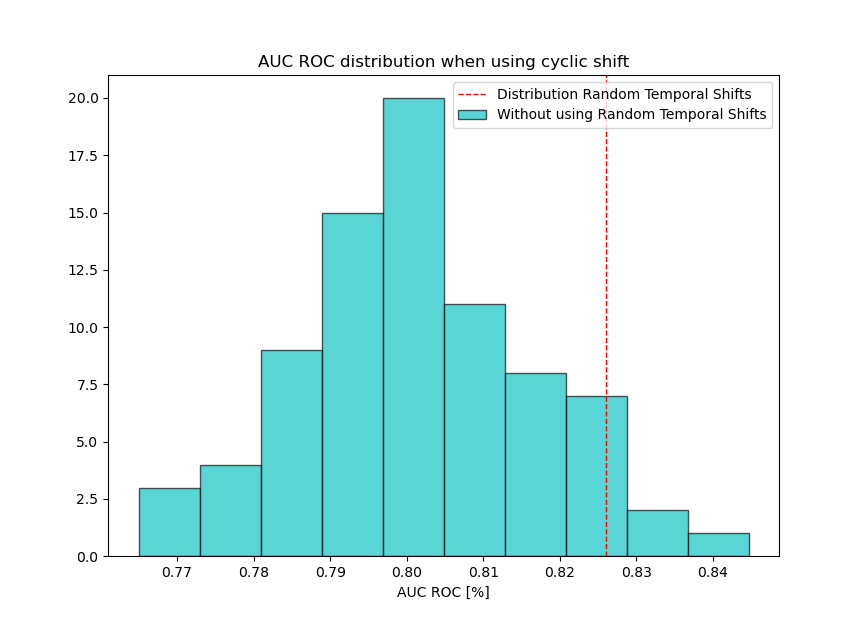
To achieve this, a (un)weighted random sampler is chosen for the evaluation set. Additionally, a cyclic shift is randomly applied to each sample. To increase consistency, we can oversample the evaluation set. Each of the 350 samples gets chosen multiple times with a random time shift each time.

This increases the computational complexity of course, so we may want to evaluate less often (increase samples per epoch).

## Tests

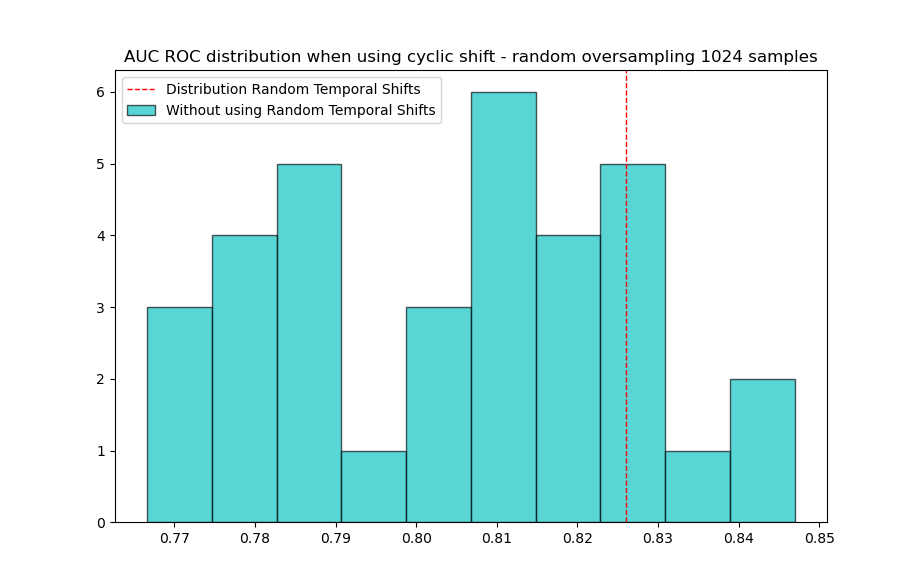
The question is: How large must the oversampling of the validation set be to get very little fluctuation. But first we need to evaluate our current situation:

As a starting point, we have a saved network, that achieved 82.3% AUCROC in the evaluation set. Since we did not do any time shifting during this evaluation, the performance is always the same during inference. We now take this network and add a cyclic temporal shift to the evaluation data. When we only use the dataset, that we have once with its 300+ samples we get a distribution:



It can be seen, that the hyperparameter search has found a setting, where the first frame of the recording (the first 224 samples = 2seconds) of the recording are rather well classified but it does not generalize this good performance to the rest of the recording. The distribution goes from 76.5% to 84.5% and has a mean and standard deviation of 80.1% and 1.5% respectively. The mean performance is more than 2% below the AUC ROC we got from evaluating without the shift.

## Adding a unweighted random sampler for oversampling

Using 1024 samples does not yield an improvement. It even makes the standard deviation a bit higher (2%) because it samples at random which samples to choose from.

Using 4096 samples the std is decreased to 1.05%. There are no more > 82% and very few < 78% inferences.

