# 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.

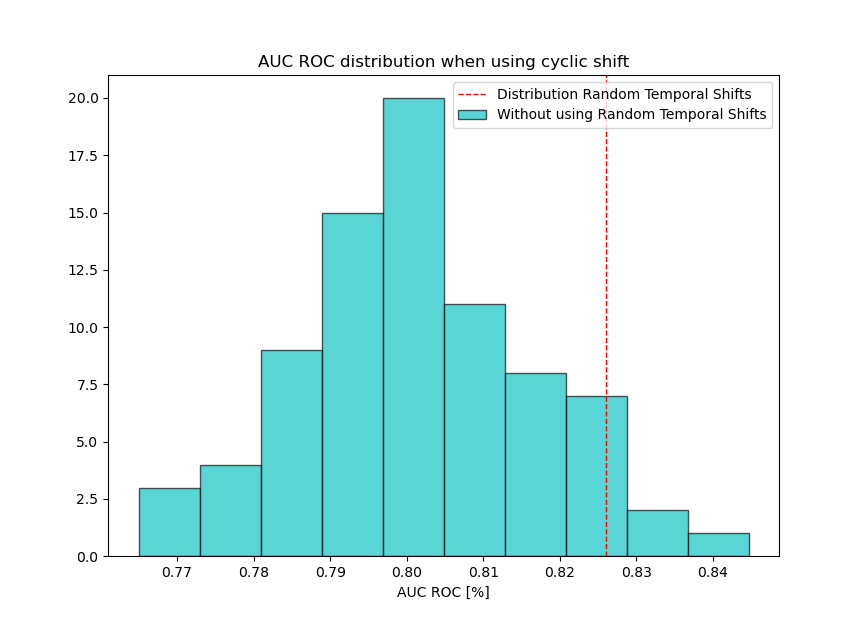
To achieve this, a (un)weighted random sampler is chosen for the evaluation set. Additionally, a cyclic shift is randomly applied to each sample. To increase consistency, we can oversample the evaluation set. Each of the 350 samples gets chosen multiple times with a random time shift each time.

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:

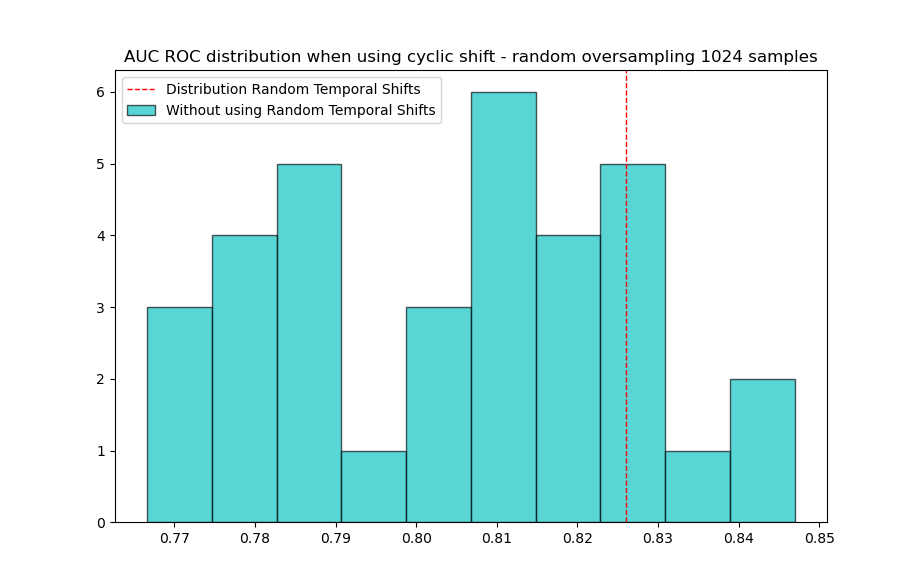
As a starting point, we have a saved network, that achieved 82.3% AUCROC in the evaluation set. Since we did not do any time shifting during this evaluation, the performance is always the same during inference. We now take this network and add a cyclic temporal shift to the evaluation data. When we only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



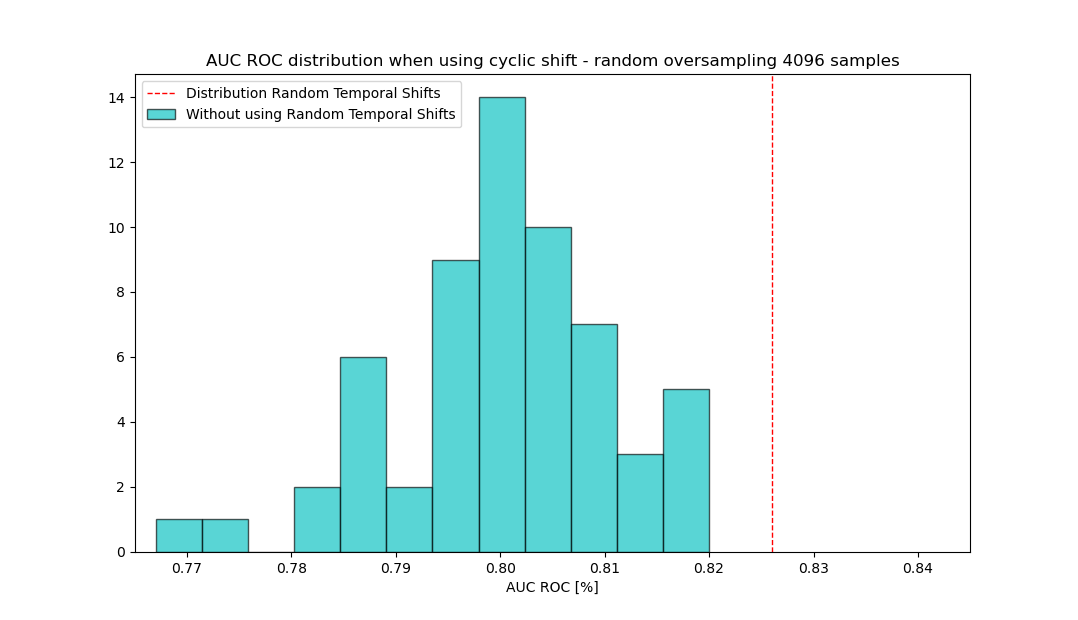
It can be seen, that the hyperparameter search has found a setting, where the first frame of the recording (the first 224 samples = 2seconds) of the recording are rather well classified but it does not generalize this good performance to the rest of the recording. The distribution goes from 76.5% to 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.

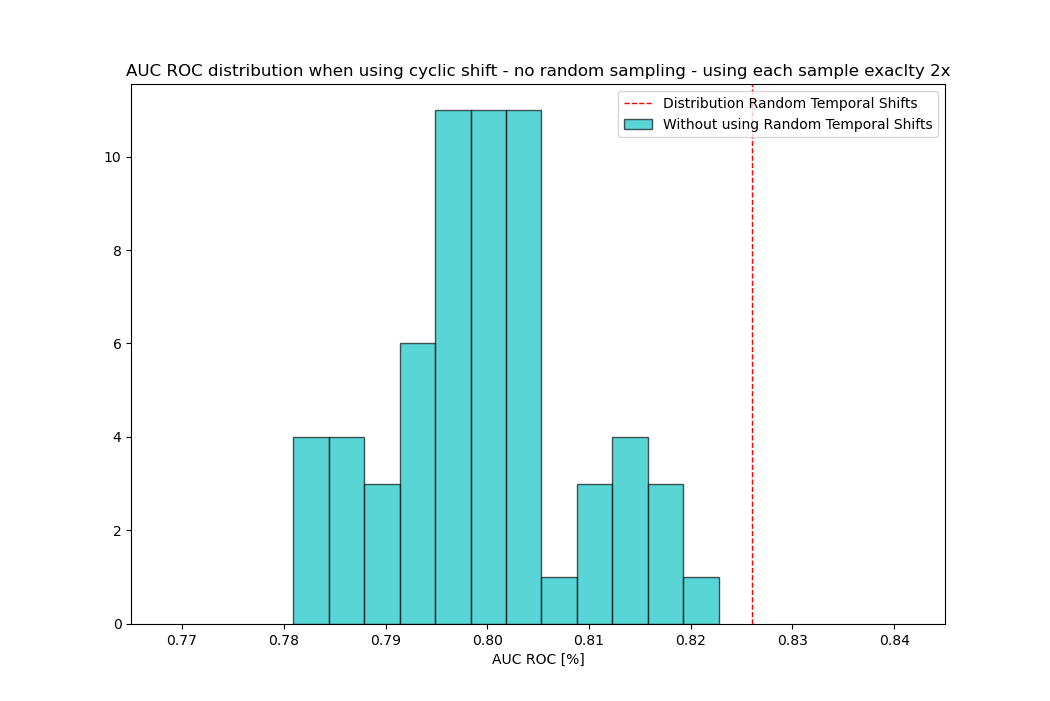


## Oversampling the entire validation set instead of random sampling:

Random sampling will result in more fluctuations which can only be counteracted by increasing the oversampling even more which again increases the computational complexity and thus the time needed for any training runs.

Instead, we now try to just use the entire validation set with random shift applied and repeat this process n times. This way, each sample will get chosen exactly n times, compared to the random sampling where some samples might be chosen 10 times while another is not chosen at all for one evaluation. The hope is, that we can significantly decrease the number of necessary samples while keeping a small variance/std.

### Oversampling the dataset 2x:

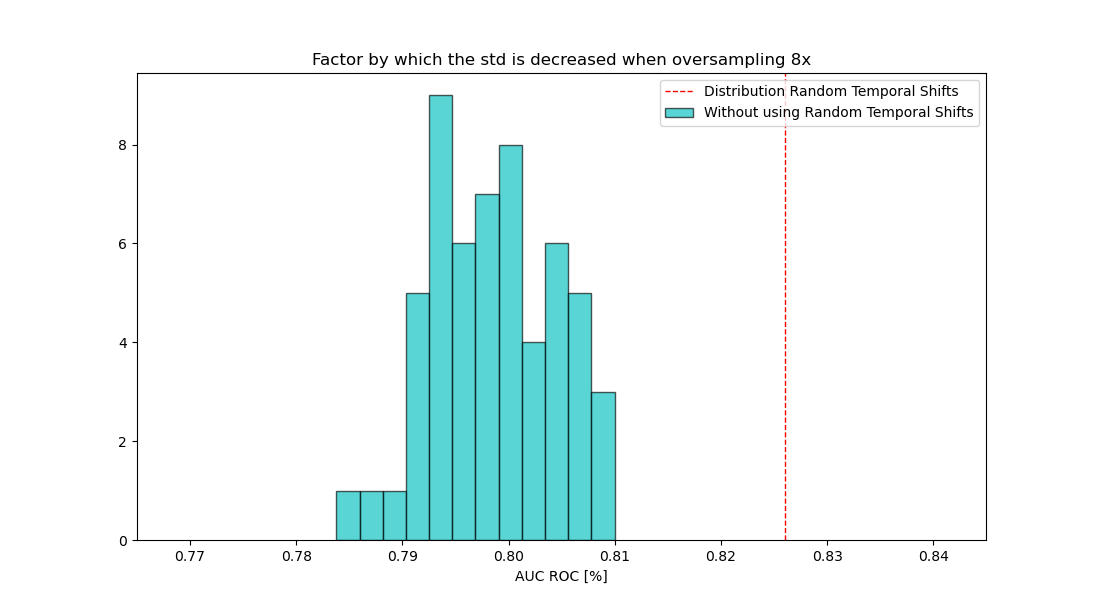
With a validation set size of ~300 samples, we use only ~600 samples for inference (compared to 4000 before) and we achieve a standard deviation of just below 1%

### Oversampling 4x:

4x300 samples make about 1200 samples. The result shows once more a significantly smaller standard deviation of 0.67% with no evaluation below 78% or above 81%.A picture containing text, screenshot, diagram, plot

Description automatically generated

### Oversampling 8x:

8x300 samples make about 2400 samples. The resulting std is only marginally smaller but still smaller (0.59%). In this case it does not seem to matter but mathematically (see below) and in a later test the decrease of the std was significant enough (from about ½ to 1/3 of the original std). So the optimal oversampling is probably somewhere between 4x and 8x. For now, I think I choose 8x just to be a bit more on the save side.

## Mathematical background (according to ChatGPT):

The standard deviation of the sample mean, also known as the standard error, is calculated by dividing the population standard deviation by the square root of the sample size. So, if you draw multiple samples and each sample contains n observations, the standard deviation of the sample mean (standard error) will be the population standard deviation divided by the square root of n.

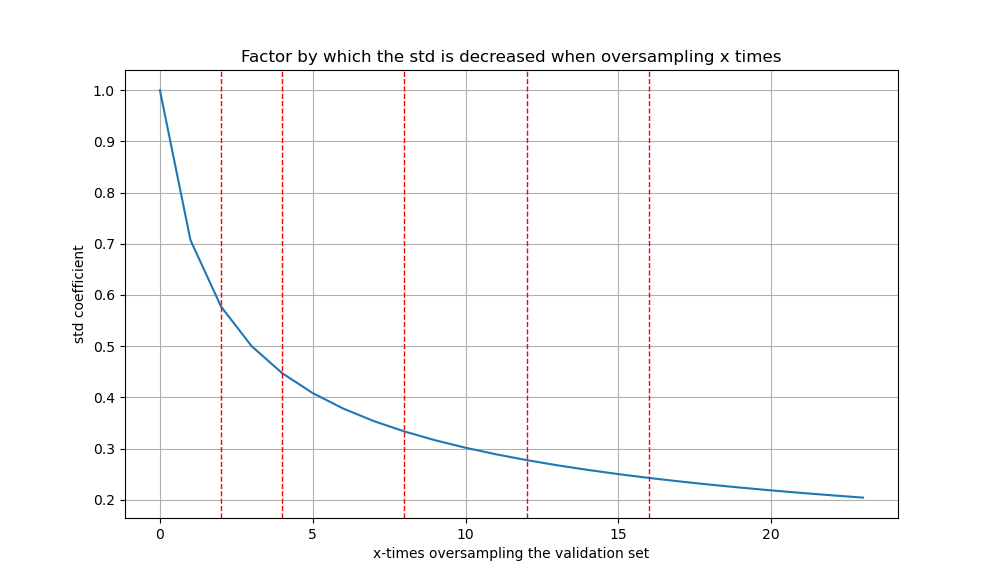
Assuming the observations are independent and identically distributed (i.i.d), the central limit theorem states that as the sample size increases, the distribution of the sample mean approaches a normal distribution regardless of the shape of the population distribution. So, as you draw more samples and increase the sample size, the resulting distribution of the sample mean will become more and more approximately normal.

The mean of the resulting distribution of the sample mean will still be 80, as mentioned earlier. However, the standard deviation of the resulting distribution, or the standard error, will decrease as the sample size increases. The relationship between the standard deviation of the population and the standard error can be expressed as:

Standard Error = Population Standard Deviation / √(Sample Size)

For averaging 2 samples: Standard Error = 1.5 / √2 ≈ 1.06  
For averaging 4 samples: Standard Error = 1.5 / √4 = 1.5 / 2 = 0.75  
For averaging 8 samples: Standard Error = 1.5 / √8 ≈ 0.53

This fits rather well with the results of our experiments: When not oversampling we get a std of about 1.5%. 2x oversampling got us 0.97% and 4x oversampling got 0.67%. It does not fit perfectly but this is most likely due to the small sample size.

Using another dataset/model we got 1.9% std without oversampling, 0.95% for 4x oversampling (which fits perfectly: 1.9%/√4 = 1.9%/2 = 0.95%) and 0.74% for 8x oversampling (1.9%/√8 = 0.67% was expected)

A reasonable choice would be 8x oversampling which results in 0.35 times the original std while using 8\*300 = 2400 samples. 95.4% of drawn samples are within +/- 2 sigma. Before (with a std of 1.5%) this meant that 95% were between 77-83. After 8x oversampling this means they are between 79 and 81% (6% vs 2% spread). For 3 sigma (99.7% within) this results in 75.5 - 84.5% and 78.5 – 81.5% respectively.