

WHO LET THE MEERKATS OUT?

IMPROVING ANNOTATION EFFICIENCY FOR
BIOACOUSTIC SOUND EVENT DETECTION
THROUGH ACTIVE LEARNING

RICHARD LINDHOLM, OSCAR MARKLUND

Master's thesis
2024:E53



LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Master's Theses in Mathematical Sciences 2024:E53
ISSN 1404-6342
LUTFMS-3502-2024
Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden
<http://www.maths.lu.se/>

Acknowledgments

First and foremost, we want to thank RISE (Research Institutes of Sweden) AB for giving us the opportunity to write this thesis under their supervision. We are most grateful for the invaluable help and feedback from our supervisors at RISE, John Martinsson and Olof Mogren. An additional thanks to Maria Sandsten at LTH (Lunds Tekniska högskola) for excellent guidance throughout this process.

Also, thanks to Niamh.

Abstract

Bioacoustic sound event detection (SED) is a critical field for biodiversity monitoring, yet the high cost of annotating data poses significant challenges. This thesis explores the application of active learning strategies to reduce the amount of annotated data required for effective model training. This is done for a segment based SED model, where batch active learning is performed by querying full audio files rather than individual segments. The data set used is created by mixing recordings from a park with vocalizations from babies, dogs and meerkats. By implementing uncertainty based querying strategies, a reduction in data demand by up to 92% is seen when compared to a baseline. These strategies query audio files with higher event density, leading to improved performance. The best active learning strategy is the proposed *top X entropy* which performs well for $X = 10$. Batch diversification using farthest traversal shows an increase in performance for other strategies, but failed to improve *top X entropy*. This shows that there is potential for assuring diverse batches and could suggest that *top X entropy* promotes diversity. The active learning results show that these methods generalise well across different datasets, highlighting their robustness and potential for broader application in other bioacoustic contexts. The benefit of active learning is shown to correlate with the frequency of events, where a higher pay-off is given in a domain where events are rare. This work advances the practicality of machine learning in bioacoustics by enhancing annotation efficiency for segment based SED models.

Keywords: Active Learning, Sound Event Detection, Bioacoustics, Querying Strategies, Annotation Efficiency

Contents

1	Introduction	1
1.1	Previous Works	3
1.2	Purpose and Problem Statement	4
1.3	Ethical Consideration	4
2	Theory	6
2.1	Sound Event Detection (SED)	6
2.1.1	Segment Based Methods	6
2.1.2	Machine Learning for SED	8
2.1.3	Advanced Network Architectures	9
2.1.4	SED Performance Metrics	11
2.2	Active Learning	12
2.2.1	Uncertainty Sampling	13
2.2.2	Query Diversification	14
2.2.3	Active Learning for SED	16
2.2.4	Performance Metrics	18
2.3	Machine Learning Fundamentals	19
2.3.1	Data Set	19
2.3.2	Training	20
2.3.3	Optimisation	23
2.3.4	Neural Networks	25
3	Data Generation	27
3.1	Method	27
3.1.1	Generating Soundscapes	27
3.1.2	Evaluation Data Set	28
3.2	Results from Data Generation	28
3.2.1	Exploring $\mathcal{D}_{0.2}^0$	28
3.2.2	Exploring $\bar{\mathcal{D}}_{0.2}^0$	29
4	SED Model	31
4.1	Summary of SED Model	31
4.2	Method	32
4.2.1	Pre-processing	32
4.2.2	Classification	32
4.2.3	Post-processing	33
4.3	Results	33
4.3.1	Visualisation of Model Output	33
4.3.2	Segmented Data	34
5	Active Learning for SED	37
5.1	Method	37
5.1.1	Simulating Active Learning	37
5.1.2	Setup	37
5.2	Segment Aggregation Methods	39

5.2.1	Comparison of Aggregation Strategies	39
5.2.2	Analysis of Queried Files	43
5.2.3	Finding a Suitable X in <i>top X entropy</i>	46
5.3	Diversification	50
5.3.1	Random Selection	50
5.3.2	Farthest Traversal	50
5.4	Generalization	54
5.4.1	$\bar{\mathcal{D}}_{0.2}^0$ (Domain Change)	54
5.4.2	$\mathcal{D}_{0.2}^{\pm 10}$ (SNR Change)	55
5.4.3	$\mathcal{D}_{1.0}^0$ (Event Ratio Change)	57
6	Conclusions	59
	References	60

List of Figures

1.1	Example: Active Learning Loop	2
2.1	Sound Classification vs SED	6
2.2	Resolution of Segmentation	7
2.3	Mel-spectrogram	9
2.4	YAMNet Architecture	10
2.5	IoU	12
2.6	Active Learning Loop	13
2.7	Example: Mean Entropy	16
2.8	Example: Median Entropy	17
2.9	Example: Mean Event Entropy	17
2.10	Example: top X entropy	17
2.11	Area Under Curve	18
2.12	Budget Reduction	19
2.13	Mean Squared Error	22
2.14	Confusion matrix	23
2.15	Gradient Descent - Global- and local minimum	24
2.16	Small Neural Network	26
3.1	Histogram: Event Length Distributions	29
3.2	Histogram: Event Length Distributions	30
4.1	SED Prediction Pipeline	31
4.2	Visualisation of the SED model output, Example I	34
4.3	Visualisation of the SED model output, Example II	35
4.4	Distribution of Event Segments in $\mathcal{D}_{0.2}^0$	36
5.1	Total IoU: Segment Aggregation Methods	40
5.2	Recall: Segment Aggregation Methods	41
5.3	Accuracy: Segment Aggregation Methods	42
5.4	Matched F_1 -score and Total IoU: Segment Aggregation Methods with Conf. Intervals	42
5.5	Queried class segments: Segment Aggregation Methods	44
5.6	Total IoU and Recall: Different <i>top X entropy</i>	46
5.7	Total IoU on Classes: Different <i>top X entropy</i>	47
5.8	Fitted Gamma Distribution of Event Segments in $\mathcal{D}_{0.2}^0$	48
5.9	Total IoU: Ensemble for <i>top X entropy</i>	49
5.10	t-SNE Visualisation - Perch	51
5.11	t-SNE Visualisation - AudioMAE	52
5.12	t-SNE Visualisation - Perch (classes)	52
5.13	t-SNE Visualisation - AudioMAE (classes)	52
5.14	Total IoU and Recall: Farthest Traversal <i>Top X Entropy</i>	53
5.15	Total IoU and Recall: Farthest Traversal Median Entropy	54
5.16	Total IoU: Domain Change	55
5.17	Example: <i>SNR</i> Comparison	55
5.18	Total IoU: <i>SNR</i> Change	56

5.19 Total IoU: Event Ratio Change	57
5.20 Total IoU on Classes: Event Ratio Change	58
6.1 Who let the meerkats out?	60

List of Tables

3.1	Summary of Data Generation	28
3.2	Distribution of Event Lengths (E_L) for $\mathcal{D}_{0.2}^0$	29
3.3	Distribution of Event Lengths (E_L) for $\bar{\mathcal{D}}_{0.2}^0$	30
4.1	Classifier-head Hyperparameters	33
4.2	Class Distribution in Segmented Data	35
5.1	Active Learning Queries	38
5.2	Data Setup	38
5.3	Queried Event Segments (%) - Iteration 3	43
5.4	Queried Event Segments (%) - Iteration 11	45
5.5	Queried Event Segments (%) - Iteration 15	45
5.6	Queried Event Segments for Ensemble (%) - Iteration 11	50

List of Algorithms

1	Pool-based Active Learning Loop	14
2	Pool-based Active Learning Loop with Diversification	15
3	Farthest-traversal	16
4	ADAM Optimisation Algorithm	25

List of Abbreviations

CE	Cross Entropy
FN	False Negative
FP	False Positive
GD	Gradient Descent
IoU	Intersection-Over-Union
MSE	Mean Squared Error
SED	Sound Event Detection
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
TN	True Negative
TP	True Positive
t-SNE	t-distributed Stochastic Neighbor Embedding

Chapter 1

Introduction

In the field of *Bioacoustics* the auditory aspects of biology is studied. Sound recordings featuring animal vocalizations, tree vibrations and a variety of other nature related sounds is the data of interest. Analysis of this data can give useful information about the ecosystem, biodiversity and animal behaviour [1].

Acoustic wildlife monitoring is relatively cheap and not invasive of the eco-system. However, the audio recordings of interest are usually long, often weeks, making it expensive and time consuming for an ecologist to analyse them [2]. Fortunately, computers can be used to automate a lot of the process and bioacoustics has benefited from recent advancements in machine-learning [1]. However, the sheer length of the data remains a problem. In order for a computer to make sense of the recordings, it needs examples to learn from. This means that a human has to analyse the data in order to tell the computer program what to make of the data. The data is thus only valuable if it is paired with human supervision. The main focus of this thesis is to study how a bioacoustic model can learn with less data by allowing it to choose what data will be most beneficial for its improvement.

Sound Event Detection (SED) relates to identifying what is happening in an audio recording by finding *events*. An *event* could for instance be a Kinkajou squealing or a Hammercopf singing. SED specifically relates to the task of detecting events by determining a start-time, end-time and specifying which *class* the event belongs to. In bioacoustics, the classes are often associated with different species. SED models can thus be used to detect the presence of species in a habitat, or count the number of vocalizations in order to estimate population size. The vocalization rates of different animal sounds in an area can be useful indicators of biodiversity, migration patterns and population size.

Machine learning paradigms like artificial neural networks and deep learning have in recent years helped automate much of the data analysis in multiple industries including bioacoustics [1]. The category of machine learning methods commonly used for detection tasks is *supervised learning*. These methods require *annotated data* which associates the data with a *ground truth*. For SED, this means that an audio recording is paired with a set of events described by their start, end and class. This annotation process is generally done manually by humans and is labor intensive. Consequently, there is a lack of large data sets with the appropriate annotations for SED which in turn hinders the development of large machine learning models [3].

Fortunately, there are machine learning paradigms that aim to ease the annotation process. The main focus of this thesis is to reduce the amount of annotated data needed without compromising on performance. This is explored through a concept called *Active Learning*. The basic idea is that the machine learning model chooses which data it needs to improve at a given task. One approach to active learning is to have the model signal which data points it is unsure about, and provide annotations for those data points. In other words we use less data to train the model at first, and then help it become more certain by focusing on its weaknesses and hopefully end up needing less annotated data in total to reach a set performance. The active learning approach has proven to be of great help in other sound related machine learning tasks such as speech-recognition [4]. In this thesis, its application towards bioacoustic SED is studied.

A Simple Active Learning Example

Imagine you are a bird-enthusiast, and want to learn to distinguish different species based on their bird song. How do you go about improving at this task?

If you are already an expert in how a robin sings, it is not that helpful to look up what it sounds like. However, if you always struggle to distinguish a sparrow from a magpie, you would probably benefit from hearing more examples of their songs. This is the core idea of *uncertainty based active learning*: that more is to be gained by learning from the instances we are unsure about.

In an active learning framework, the learner *queries* datapoints it wants to learn from based on some *querying strategy*. This strategy could be based on uncertainty as we previously discussed, but could also be based on similarity between data points.

Let's put this into context with our hypothetical example. Imagine you have access to a large data bank of bird songs, the problem is that the *labels* are missing meaning you cannot tell which species of bird is singing. Fortunately you have a friend who is an acclaimed ornithologist, specialised in bird songs. The problem is that she is very busy meaning you can only ask her about three song recordings a day. This hypothetical scenario is quite similar to how active learning works in reality. There is an *oracle*, your friend, who can help provide labels or *annotations* for the data points you are interested in learning from. You are allowed to iteratively ask queries, often multiple at once in a *batch*, so that you can learn from these examples. Once you have studied the examples, you get to make new queries based on your current knowledge.

One day you find three bird songs that you are clueless about, but they sound almost identical and you can therefore assume they come from the same species. You quickly figure that you don't need to ask your friend about all three, as one is enough to give you the sufficient knowledge about all three. Instead you can query one of the three similar bird songs, and choose two other songs for your daily batch. By doing so, you improve at multiple species by opting for variety in the batch. This is an example of *diversifying* queries, and it makes you think about how to do this in the most effective way.

One day, the data bank is compromised by a Goldfinch enthusiast and is now filled with 95% Goldfinch songs. There is now an extreme *class imbalance*. Does this affect how you should go about your learning process?

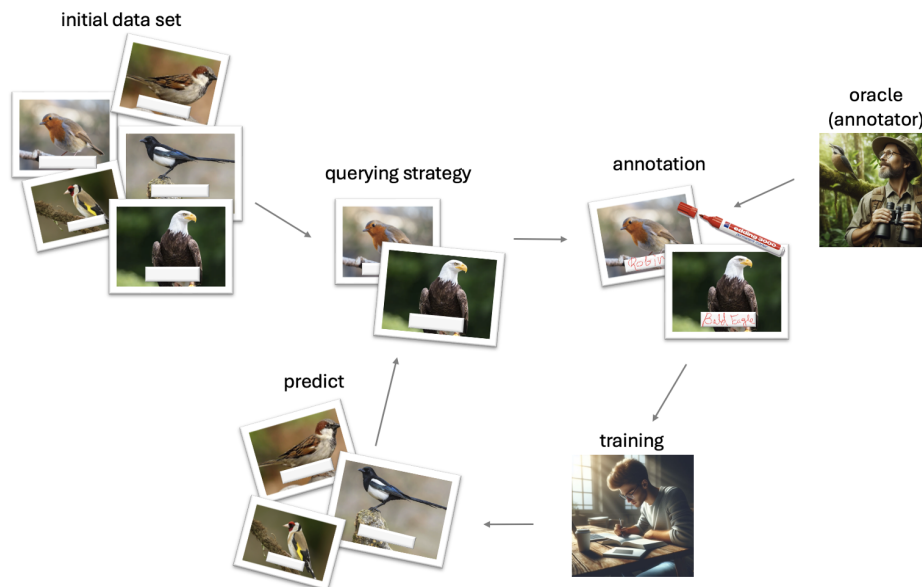


Figure 1.1: **Example: Active Learning Loop Example** - Visualisation of an example of active learning and its fundamental components.

This example illustrated the fundamentals of active learning. We have discussed many key concepts; queries, batches and the oracle. We have mentioned that uncertainty can be a strategy for making queries. We have also posed the question about how imbalances in data and diversification might impact learning curve. Whilst being an illustrative and simple example, much of what has been discussed will be relevant in this thesis when teaching a computer how to detect bioacoustic events.

1.1 Previous Works

Research on the topic of active learning for SED exists, but is scarce. The 2020 research paper by Shuyang et. al claims to be the first on the matter [5]. In this paper, the proposed active learning strategy was proven to be very helpful in reducing the amount of annotation needed. With only 2% of the available data, their active learning framework was able to get results that were close to that of a model trained on all the available data. This paper proposes a novel active learning framework that queries annotations for data points that are classified differently by two classifiers, and are dissimilar from data points that are already classified. This approach is called mismatch-first farthest traversal.

This thesis differs in methodology compared to [5], as the active learning strategies and SED models are different. For instance, the active learning queries in this thesis are limited to audio files, whereas in [5] queries are made for only predicted events. This difference stems from the difference in how the SED models are designed. How queries are selected also differs, as the work presented in this thesis is more closely related to uncertainty based active learning. In this thesis the farthest traversal principal is studied in a new context, inspired by the work in [5].

Other active learning strategies have been studied for SED. In [6] an uncertainty based strategy is tested, which outperforms a baseline strategy that uses random querying. Whilst this paper also studies uncertainty based active learning for SED, uncertainty is defined and used to make queries differently compared to experiments in this thesis.

A common way to construct an SED model is to divide the audio into short segments, and classify each segment. Such a model is called a *segment based* SED model, and is used in this thesis. Uncertainty based active learning for such a model requires that queries are selected based on many predictions/uncertainties. This means that many uncertainties need to be *aggregated* into a measure that is representative of the entire audio file. No research has been found on the topic of active learning for a segment based SED approach, or how to aggregate uncertainties across an entire file. However, there exists research on active learning for similar methods when studying object detection in images, where uncertainties for smaller segments are aggregated to represent the full image. Two such papers are [7] and [8]. Whilst these analyse images opposed to audio, they are still based on similar ideas that can be used to build an active learning framework for SED. In [7] they found that aggregation through summing uncertainties was the best active learning strategy, as this favours images with many objects. In [8], an aggregation strategy which is meant to address diversification issues is tested and successful in improving performance. This thesis aims to see if these result transfer to a completely different domain, but where the model is built based on similar reasoning.

There exists research on a topic called *batch active learning*, two examples being [9] and [10]. This concept is related to optimally selecting a *batch* consisting of multiple queries. This usually involves diversifying the batches by opting for queries that are different from one another. In this thesis, batch active learning is studied as multiple files are always queried at once. [9] adds noise to the query selection, which is done in this thesis as well in an attempt to diversify batches. With that said, the context of SED, and how noise is added is different in this thesis.

The training of a segment based SED model is very similar to building a sound classification model. Active learning for classification tasks across multiple domains has been studied before in [11], [12] including bioacoustic sound classification [13]. These efforts generally show that active learning is beneficial in reducing the number of annotated data points. Whilst SED isn't only about classification, these are still promising results for the prospect of using active learning for SED.

There exist other efforts to reduce the amount of annotated data for SED, but using techniques that aren't directly connected to the concept of active learning. Few-shot learning is one such approach, which has also been studied for SED in a bioacoustic context. Few-shot learning has been used to successfully detect animal vocalizations with as few as five annotated samples per detected class when using a SED model based on deep frame-level embeddings of audio recordings [14]. Few-shot learning methods often utilise pre-trained machine learning models which are used in a new domain. This method is also used in this thesis to build the SED model with less data and training. Few-shot learning should not be seen as an alternative to active learning efforts, but as a complementary method to even further reduce demands for annotated data.

1.2 Purpose and Problem Statement

One limitation in bioacoustics is the lack of annotated data and this thesis aims to find a solution to this problem. This is done by studying active learning, specifically for bioacoustic SED. The purpose is to make the field of bioacoustics more accessible by making the practical application of machine learning methods more realistic.

The goal of this thesis is to improve the annotation efficiency of bioacoustic SED. Active learning frameworks proven successful in other domains are applied for SED, together with a few novel methods to successfully design a good querying strategy.

The main focus lies in creating a successful querying strategy when using a segment based model, and determining how the parameters of the strategy affects performance and what type of queries are made. Batch active learning is studied in order to see if attempts to diversify batches are successful. We are also interested in how active learning results depend on the data set. To address these objectives, the following research questions are posed:

- Can uncertainty based active learning reduce the annotation cost for a SED model?
- What is the best way to aggregate segment uncertainties in order to benefit performance?
- Is it possible to improve uncertainty sampling through diversification of queries?
- How does active learning depend on the underlying data distribution?

In order to answer the research questions, many active learning methods are tested by simulating active queries. The different querying strategies are compared by analysing detection metrics which consider performance in terms of classification and temporal placement of events. This is compared to a baseline which is based on random queries, which resembles how a model would receive data without active learning.

The data sets studied are audio files created by mixing foreground audio in the form of events with background audio in the form of noise. The audio is *monophonic*, meaning that no events overlap each other. As overlapping events would require a more complex model, this restriction is set to emphasise active learning rather than SED. As the data set is synthetic, it is easy to study how different parameters may affect how effective active learning is. One such parameter is the signal-to-noise ratio, meaning the robustness of the active learning frameworks can be examined. Other parameters include how frequent events are. The synthetic data allows for testing of many different underlying distributions, but since the time period of this thesis is limited, restrictions must be made. Therefore only three event classes are considered, babies, dogs and meerkats. These are mixed with background noise from a park. Many data generation parameters are kept fixed as testing many combinations of parameters is time consuming. Each audio file is assumed to cost equally much to annotate, no matter the amounts of events occurring in them.

1.3 Ethical Consideration

Machine learning and audio monitoring presents a broad spectrum of ethical considerations that demand careful attention.

- Training machine learning systems requires significant computational power when trained. This contributes largely to the environmental footprint. It is usually the training of models which requires a lot of energy, and in this thesis we don't train any large models from scratch which saves a lot of energy. As active learning aims to reduce the amount of data needed, it can also help alleviate the amount of training that is needed. We hope that active learning efforts can help make machine learning more computationally efficient. We also hope that bioacoustic models will become more accessible and thus override the environmental impact with its social importance.
- There are ethical concerns regarding the data used in the training of many models. Audio monitoring might be intrusive, if done wrong. The data used in this thesis does not intrude on anyone's privacy.
- Transparency is a fundamental principle in ethical machine learning. Understanding how algorithms make decisions and being able to trace back those decisions is crucial for ensuring accountability and trustworthiness. It's also important to be transparent with the data used and how it is being handled.
- Sometimes, jobs are at risk of being replaced with automated solutions. With bioacoustic monitoring however, the task of analysing months worth of auditory data is intractable for an ecologist. The type of model studied in this thesis should be seen more as a tool for ecologists to better understand an ecosystem. Their expertise is still very much needed to make to train the model and make sense of its predictions.

In conclusion, ethical frameworks and guidelines for machine learning should be developed alongside the technological breakthroughs. This is to promote a responsible and beneficial use of machine learning for all.

Chapter 2

Theory

In this chapter, we present the necessary theory for an easy understanding of the experiments in this thesis. This chapter begins with an explanation of how SED models are built in 2.1. This is followed by a section on active learning 2.2, which introduces the theoretical concepts that this thesis is centered around. In both of these sections a solid understanding of machine learning concepts is assumed. If the reader needs an introduction to machine learning fundamentals, please begin with reading 2.3. This section explains the most fundamental concepts of machine learning, meaning if the reader is familiar with these, feel free to skip section 2.3.

2.1 Sound Event Detection (SED)

In *Sound Event Detection* (SED) the objective is to know what is happening in an audio signal, and where it is happening. Contrary to Sound Classification, also known as *tagging*, a SED system does not assign labels to entire clips of audio, but instead attempts to assign labels only to the parts of the clip where an event occurs. This means that an annotation, and prediction, consists of three parts; a discrete label of *what* sound the event contains, and two continuous labels in the form of a start-and end time of *where* the sound occurs. These type of annotations are referred to as *strong labels*, contrary to *weak labels* which solely assign classes to entire files [3], see figure 2.1 for visualisation of the difference.

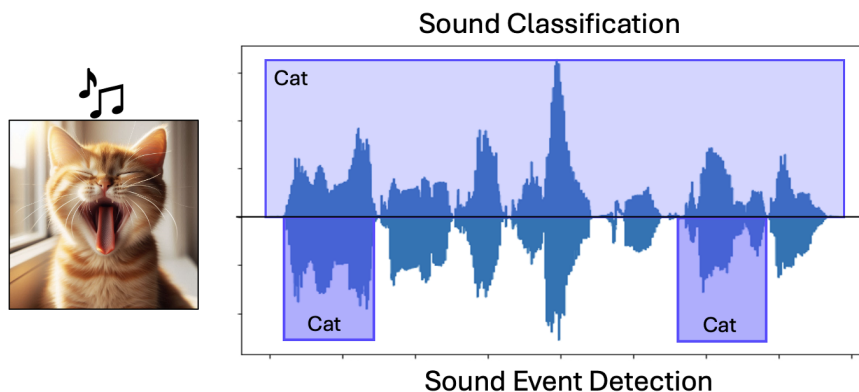


Figure 2.1: **Sound Classification vs SED** - The fundamental difference between sound classification and SED, where SED takes the time of the events into account.

2.1.1 Segment Based Methods

A common approach to perform SED is to use a *segment based method*, where the audio being studied is divided into short *segments* [3]. The size of these segment are predetermined and may overlap with each other. Each segment is analysed separately and weak labeled, where adjacent segments with the

same label approximate a full event, the strong label. The size of these segments determines the resolution, and thus the precision in the time-domain, of the prediction. This is due to that the previous domain, being the entire \mathbb{R}^+ , has been reduced to a discrete set of time labels, being the start- and end times of all segments. Depending on how the machine learning model is constructed, there could be a trade-off between computational efficiency and the loss in resolution. Further, a short segment size allows for high-resolution and detection of shorter events, but less information is provided to the model, which can also affect accuracy. On the other hand, segments that are too long can contain multiple events, miss short events and reduce resolution. The resolution is often set to be appropriate for the length of the ground truth events [3].

Furthermore, for a model to train on these segments in a supervised manor each segment has to be assigned a corresponding true label in the training phase. A naïve way of modifying the previous strong labels into segments is just to assign the segments an event label if the sounds event exists within in range of the segment [3]. If one would define a segment as an event only by this fact, that the segment contains a part of an event, the new start- and end time of the true event in the new domain can differ as much as the length of a segment at the start and end of the event. Figure 2.2 shows the best- and worst-case scenario of how the resolution is affected by segmentation. It is apparent that the resolution can get reduced when switching to a segment based domain, as well as how the segment size might affect this. Instead of basing the segment labels on if an event overlaps with the segment, one can set a threshold for how much of an event needs to overlap with the segment in order for the label to be assigned.

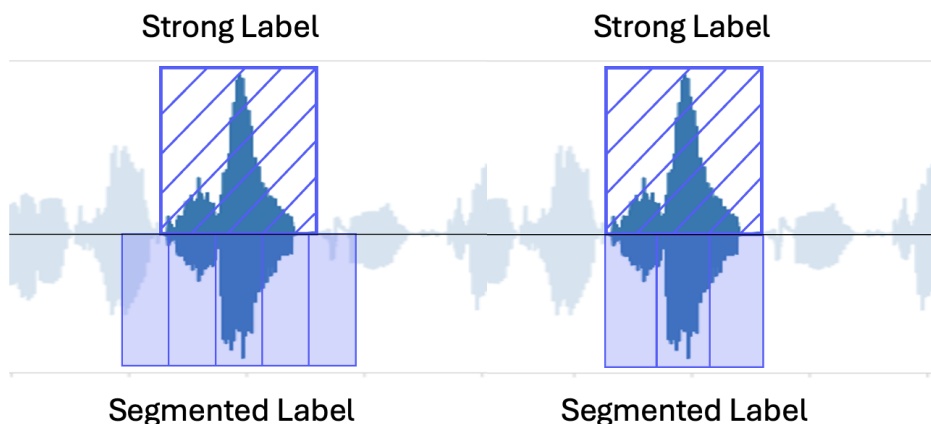


Figure 2.2: **Resolution of Segmentation** - (right) A hypothetical best-case scenario with no resolution loss when segmenting the strong label into segments. (left) A hypothetical worst-case scenario with full resolution loss when segmenting the strong label into segments.

We have explained how one can go from event labels to segment labels. We also need to be able to go in the opposite direction. Given class predictions on each segment, this needs to be processed into events. One way of doing this is to simply *merge* adjacent segments with the same predicted label into one event. This means that a single wrongfully predicted segment can ruin and contaminate an entire event. To handle these outliers, and add temporal context to the prediction a *filter* can be used before merging.

Filtering

The objective of the filtering is, amongst other things, to remove noise in the form of outliers. It also adds temporal context to the segment prediction as neighbouring segments will consider each others output. One filtering method that is effective in removing outliers is the *median filter*. This filter requires a predetermined *kernel size*, ks , i.e. the number of consecutive segments that contributes to the classification of the segment being filtered. Lets say the kernel is of size $ks = 3$, then the median value of three contributors will determine the label of the segment being filtered. When processing a segment,

the three contributors are the previous, the current and the upcoming segment. This implies that in order for a segment to be assigned a class, it must be the class with the highest median over ks segments.

We can describe the filtering in somewhat more technical terms. Assume we have a softmax output of a neural network with output dimension C (classes) for W segments, i.e. $\hat{\mathbf{y}} \in \mathbb{R}^{C \times W}$. The median kernel is applied on the temporal (segment) dimension of each node, which gives the filtered output $F_{ks}(\hat{\mathbf{y}})$. For each segment in the filtered output, the argument of the node with highest value corresponds to the predicted class. Assuming no other post-processing of the segment predictions, this gives the classification output. \mathbf{c}_{pred} is a vector of output nodes for all segments \mathbf{w} in the file. $\hat{\mathbf{y}}(\mathbf{c}, \mathbf{w})$ is the raw softmax output for all segments, that is filtered through F_{ks} . The argmax thus return a vector \mathbf{c}_{pred} that corresponds to the predicted class for each segment in \mathbf{w} .

$$\hat{\mathbf{y}}(\mathbf{w})_{SED} = \arg \max_{\mathbf{c}_{\text{pred}}} F_{ks}(\hat{\mathbf{y}}(\mathbf{c}_{\text{pred}}, \mathbf{w})) \quad (2.1)$$

This is the final classification output, which can be formatted into event labels by merging neighbouring segments of same class. Events that are too short to be plausible are often discarded [3]. This means we can predict events, consisting of a start time, end time and class, which can then be compared to the strong labels in the annotated data.

2.1.2 Machine Learning for SED

In this section we go through the theory behind analysing sound, focusing on methods used in this thesis. This mainly includes more advanced machine learning concepts, but also traditional signal processing techniques that remain relevant to the field of audio analysis.

Signal Processing

The human ear has a way of distinguishing relevant sounds from noise in its surroundings. In a crowded room humans manage to maintain a conversation by filtering out all irrelevant sound. However, if the background noise is too loud, humans might start to struggle with keeping the conversation afloat. In signal processing and machine learning, the situation is analogous. It is harder to distinguish relevant signals from the noise if the noise is of high intensity. This is also the case when building a bioacoustic SED model. It will be harder to find a dog barking in the audio if there is a lot of wind.

The *Signal-to-noise ratio* (SNR) is defined as the ratio between the power of a wanted signal, animal vocalizations in our case, and the power of the noise, the background sound. In the decibel scale this can be written as:

$$SNR = 10 \cdot \log_{10}\left(\frac{P_{\text{signal}}}{P_{\text{noise}}}\right) \quad (2.2)$$

In the SED case this is of the utmost importance as models can vary in performance depending on the SNR. It is important to check if results are good for different SNR values as this could mean that the model is robust and might be able to maintain performance when changes of microphone placement or weather occurs.

When analysing audio, raw acoustic data is not usually used because of its high dimension and hard interpretation. Instead it is common to convert raw audio signals into a spectrogram as this is more intuitively analysable, and when analysing sound in the human hearing range a so called *Mel-spectrogram* is often used. The standard spectrogram is a visual representation of the frequency spectrum of a signal as it varies with time. It is computed by dividing the audio into overlapping segments and then the Fourier transform is applied on each segment which results in different magnitude spectra over time. The Mel-spectrogram differs from a standard spectrogram in that it uses the Mel scale for the frequency axis. The Mel scale is a perceptual scale based on human hearing, which reflects the non-linear way in which humans perceive sound frequencies. Mathematically, the Mel scale is defined [3] as:

$$mel = \frac{1000}{\log 2} \cdot \log\left(1 + \frac{f}{1000}\right), \quad (2.3)$$

where f is the frequency in hertz. This transformation compresses the higher frequency, while maintaining a finer resolution in the lower ranges. This simulates the human ear’s sensitivity. The mel-spectrogram is a 2-dimensional representation of the data that can be used as input for machine learning models, including SED models. The matrix representing the mel-spectrogram can be visualised as a heatmap as seen in figure 2.3. With our data now being ordered into 2-dimensions, architectures for classic image analysis can be used for analysing the audio.

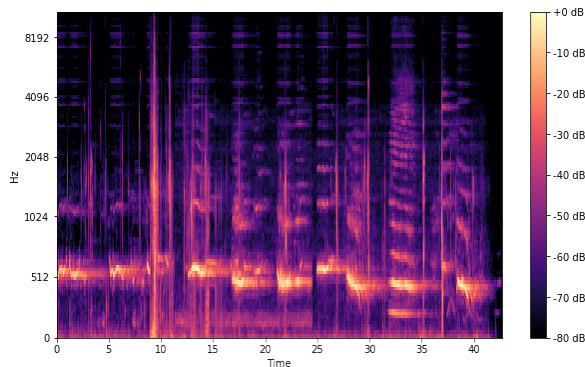


Figure 2.3: **Mel-spectrogram** - An example of a mel-spectrogram in the form of a heatmap. Strong amplitudes for different frequencies can be distinguished at different points in time.

Transfer Learning for Neural Networks

Transfer Learning refers to transferring knowledge from one domain to another. This often means training a model on a task and then reusing it on a different, preferably related task. One salient use of transfer learning is the utilization of *feature embeddings*. These are representations of the input data, often taken from an intermediate layer of a neural network [15]. A common, and simple, way to produce embeddings is to train a neural network on a classification task, and then simply remove the final classification layers. We refer to these classification layers as the classifier-head. The remaining network does no longer output class probabilities, but instead outputs a representation of the data that is useful to determine the class [1]. As the intermediate layers often are of smaller dimension than the input, the network is forced to find a smaller representation of the input data, i.e new features, that still contain enough relevant information to classify the sample. We will call such a representation an embedding of the data point.

One great advantage here is that large and well annotated data sets can be used for *pre-training*. The pre-trained model can then produce embeddings on new data that can then be used to train a classifier-heads on a new domain. In a sound analysis setting, this means that instead of training a classifier from scratch on acoustic features, one can embed these using a pre-trained model and then use these embeddings to train a small classifier model [1]. In bioacoustics, data is more common for certain taxonomic groups. Whilst there is a lot of data for common bird species, endangered species aren’t as well represented in data [15]. By utilizing a pre-trained model, that effectively produce embeddings with relevant features describing the raw audio or spectrogram, less training examples are needed.

2.1.3 Advanced Network Architectures

In bioacoustics, the standard neural network architecture is outperformed by more advance architectures, including different *convolutional neural networks* (CNN) and *recurrent neural networks* (RNN) When building acoustic models it is common to deploy pre-trained networks that use well-known architectures, such as ResNet, VGGish or MobileNet, by utilizing their embeddings through transfer learning. These architectures are usually variations of CNNs, RNNs or both [1]. In this project, some pre-trained models based on these architectures are used. In the context of SED, these types of models can be used for classification or to encode data into a lower dimension.

CNNs have *convolutional layers* which contain convolutional filters rather than normal weights. Each kernel/filter in the layer is convoluted with the input and the output is passed to the next layer, which usually applies some operation that reduces dimensionality, such as max-pooling. The weights in the CNN are trained in a similar way to a normal neural network. The difference is that some weights are located in filters. This allows for weight sharing, as the same weights slide across the spatial or temporal data as a part of the convolution operation [16]. RNNs on the other hand feature *recurrent edges*, or *feed-back connections*. These are often used to model sequential data. CNNs and RNNs are able to manage high-dimensional data without an intractable number of weights. When analysing audio, CNN classifiers are also invariant to time-shifts in the input data which is a good property to have. [1]

YAMNet (MobileNet Architecture)

In this project, a pre-trained network called YAMNet plays a fundamental role as it is used for transfer learning in the SED model. YAMNet is an open-source deep neural network for sound classification developed by Google. It is a classifier trained on 521 different labels/classes based on Google's own audio data set *AudioSet*. This data set contains over two million 10-seconds-long Youtube audio clips annotated by humans, making up about 6 thousand hours of audio data [17]. The architecture of YAMNet is based on the concept of depthwise-separable convolution inherited from the *Mobilenet_v1 network*. Compared to classic convolutions, the depthwise-separable convolution prevents possible overfitting by initializing less parameters. This is done by separation of dimensions, both spatial and depth, of the convoluting kernel. This drastically reduces the number of parameters and multiplications performed and thus the complexity and computational time [18]. The network takes an audio file in a single-channel 16kHz sampled waveform as input. 0.96 second segments are then extracted, with 0.48 second overlap, and processed separately. Note that the pre-trained YAMNet model pre-processes the data by transforming it to a mel-spectrogram. The YAMNet network can be used for transfer-learning as a feature extractor, if the final classifier-head is removed, see figure 2.4 for the full architecture. The YAMNet embedding output is a 1024 long vector for each segment.

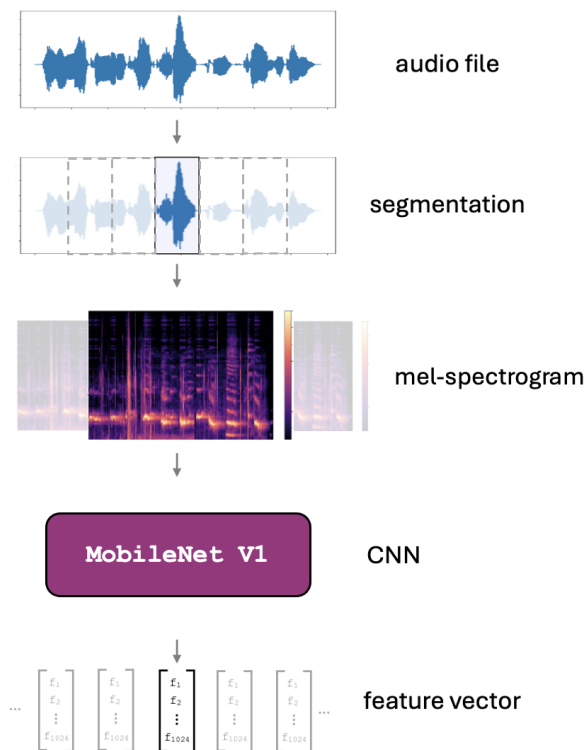


Figure 2.4: **YAMNet Architecture** - The full architecture of YAMNet as a feature extractor. Each segment in the waveform is embedded into a feature vector, which produces a time-series of embeddings.

Perch and AudioMAE

In this project the embeddings of two other models are used as a way to compare entire audio files. Whilst these embeddings aren't used for classification or SED, they are used as an attempt to improve active learning performance. These are explained in short below.

Perch is a model development by Google that uses the EfficientNet B1 network architecture, which is a pure convolutional model which proposes a network scaling/balancing method designed to be suited for the amount of computation that will be done [19]. Perch is trained on all the bird songs in the Xeno-Canto data set [15]. The data set includes more than 10,000 classes. Perch embeds longer sequences of data compared to YAMNet, 5 seconds sampled at 32 kHz. The embeddings are of length 1280. The main focus of the Perch model was to produce quality embeddings [20]. In [15], Perch was deemed to be one of the best models for enabling transfer learning for bioacoustic sound classification, out of the models that were compared. It was the most consistent model out of the bunch, proving that its embeddings extract useful information about the contents in the audio that can be used to classify audio from a different domain.

AudioMAE is a model developed by Meta. It is based on the model *Masked Auto Encoders* (MAE) model which is a vision-transformer [21]. It is trained to encode a spectrogram with some parts masked out, in order to then decode the encoding and reconstruct the input. AudioMAE uses a stack of standard transformers [22] as an encoder. The model is trained on a variety of data sets, including AudioSet [17]. AudioMAE is a successful model for sound classification and outperformed the state-of-the-art models at the time of publication [23]. However, when used for transfer learning in [15], it did not perform as well as YAMNet or Perch. The version of AudioMAE used in this project embeds 10 seconds of audio into a vector with 768 elements.

2.1.4 SED Performance Metrics

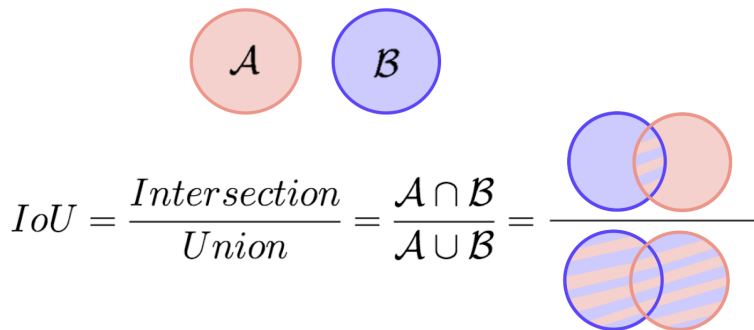
Studying the performance of a SED model is not trivial. Standard performance metrics, such as the *mean-squared-error* (MSE) or accuracy are harder to define as SED needs to consider both the ability to determine the correct class but, also the temporal placement of events.

Consider a model used for multi-class classification problem in bioacoustics using the segment based method described in section 2.1.1. The accuracy of segments predictions reveals useful information about the models classification ability. However, this metric doesn't necessarily capture the performance in the time domain, and isn't based on the final SED output, see equation (2.1).

New metrics that are engineered for SED are needed. The two properties that need to be emphasised and rewarded are the models classification ability and how well timed the events are in relation to the ground truth. As previously mentioned, there is a trade-off, where the smaller segment sizes result in better resolution, but less information is provided for each prediction.

The final prediction $\hat{\mathbf{y}}_{SED}$ consists of event labels. For an event to be considered a successful prediction it has to match the true event in both the time domain, where start- and end time has to be correct and the correct label has to be assigned to the event. A metric that fulfills both criteria is the *Intersection over Union* (IoU). The *intersection* is the part that two sets share, in our case it is the time period that both the predicted event and the true event cover. The *union* is defined by the total area covered by both sets, this equals the sum of the intersection and the unique parts of both sets. Taking the ratio between these two measurements gives us the IoU metric, which can be considered for comparing predictions and true events of the same class. Figure 2.5 shows two sets, \mathcal{A} and \mathcal{B} , as an example of the metric IoU, along with its definition.

IoU can be used as a metric for events of the same class, or by setting a threshold for correct predictions. When counting *true positives* (TP) i.e true events, it might be reasonable to set a minimum threshold ϵ for IoU. If a 10s event is predicted to only be of 1s, it might not be fair to call this a TP. If the IoU of a predicted event and true event with the same class is larger than this threshold, we count this a *matched event* or a TP. Note that there can only be one matched event per ground truth event.



$$IoU = \frac{Intersection}{Union} = \frac{\mathcal{A} \cap \mathcal{B}}{\mathcal{A} \cup \mathcal{B}} = \frac{\text{Intersection of } \mathcal{A} \text{ and } \mathcal{B}}{\text{Union of } \mathcal{A} \text{ and } \mathcal{B}}$$

Figure 2.5: **IoU** - The definition of IoU along with an example using two sets, \mathcal{A} and \mathcal{B} . The second fraction shows the mathematical symbols used in the field of set theory for both *intersection* and *union*. The last fraction shows a visual interpretation of the two metrics, where the striped area is the area of interest.

If there are multiple predicted events for one true event, all but one will be considered *false positives* (*FP*).

In the occurrence of multiple predicted events corresponding to a true label, the events are handled as a bipartite graph. By using Hopcraft-Karp-Karzanov algorithm, the maximum-cardinality matching can be determined and decides which events that should represent the true event. Basically, this means that the most "suitable" event is matched with the true event, where the rest will be considered as *FP*, and wrongfully classified. If no predicted event is assigned to a true event it is considered as a *false negative* (*FN*).

By counting *TP*, *FP* and *FN* like described, using bipartite matching based on some threshold, a F1-score (2.14) can be determined in the event domain and used as a performance metric for the SED model. This F1-measure has some temporal dependency, as it uses the matched events criteria. IoU can also be used as a metric on its own as it describes how well the predicted event actually covers the true event. However, a decision needs to be made about how IoU should be measured across an entire data set of many files. One strategy is to imagine that all the files are added into one long file, and then calculate the IoU. This variant of IoU will be referred to as the total IoU. The total IoU gives a good estimate of the total performance in predicting both class and time correctly. [24]

2.2 Active Learning

A SED model needs data to train on in order to accurately find events and classify them. Depending on the domain that is studied, data can vary a lot in accessibility and how expensive it is to annotate, both in time and money. Annotations for SED generally come with a larger annotation cost than sound classification, as they require start time and end time. In the context of bioacoustics, some events might be scarce, meaning that a lot of data needs to be processed and annotated in order to find sufficient training data to support these events. This has led to a scarcity of large annotated data sets. Active learning is designed to bypass this problem, by finding data points that should be annotated in order for the model to gain the most in performance. If successful, this means that fewer annotations are needed to reach a set performance. In active learning it is common to speak of a labelling *budget*, which can be measured in different ways. This can for instance refer to how much time in total the annotator will spend annotating, or how many data points the annotator annotates. The goal of active learning can thus be summarised as producing good results with a small budget by carefully selecting the files that are annotated.

Active learning attempts to aid machine learning models that are trained on data which is expensive to annotate. The main idea is to allow a model to choose which data to learn from in order for it to perform better [4]. Active learning usually works by training a model on an initial pool of labeled data

that is usually quite small in relation to the amount of unlabeled data that is available. The model is then allowed to ask for annotations by selecting a *batch of queries* consisting of unlabeled data points. These queries are annotated by an oracle, an entity which assigns ground truth labels to the data. The selection of which data entries to query should be strategically chosen, to best improve performance. This process is repeated iteratively as seen by the schematic figure 2.6.

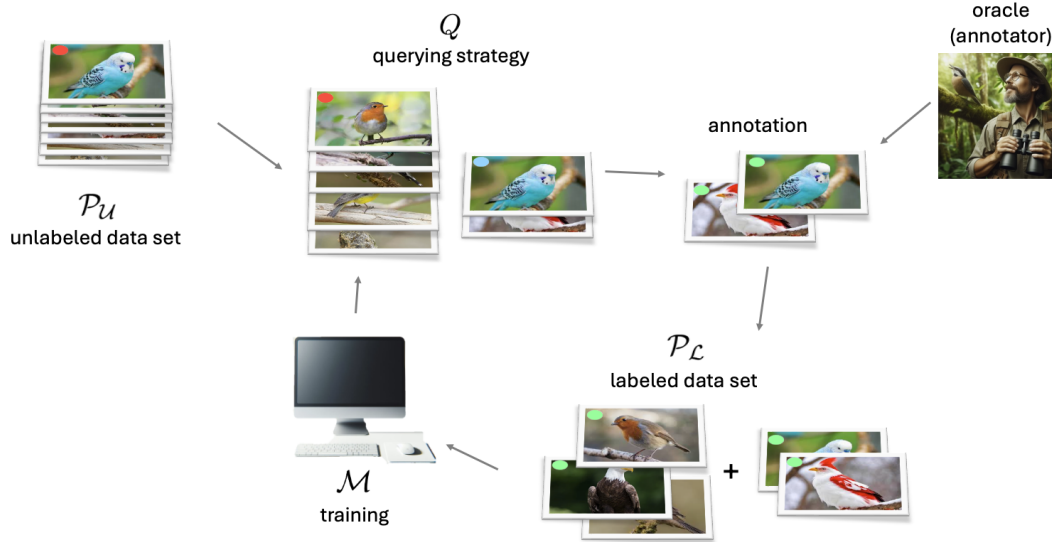


Figure 2.6: **Active Learning Loop** - A visualisation of an active learning loop with all its components and steps. The colored circle notations are as follows. Red is unlabeled data points. Blue is data points being queried. Green is labeled data points.

Pool-based active learning is a common approach to active learning. This approach selects which data entries are to be labeled by the oracle from a large pool of unlabeled data \mathcal{P}_U . Entries that have been labeled by the oracle belong to a pool of labeled data which is denoted \mathcal{P}_L . Which instances are queried from \mathcal{P}_U is commonly decided based on an uncertainty measure, favoring predictions of high uncertainty [4]. This approach is called uncertainty sampling and elaborated below.

2.2.1 Uncertainty Sampling

Uncertainty sampling queries data points where the model has low confidence in its predictions. This strategy targets data points where the model is ambiguous, meaning it helps the model where it struggles. This strategy also makes intuitive sense in multiple ways if we consider why a model might be uncertain. If a class is under-represented in the data set, it might lead to uncertainty in the model prediction. In this case, uncertainty sampling can help make classes more equal in the queries. On the other hand, if a class is very difficult to predict, the model would benefit from seeing disproportionately many queries from this class, which uncertainty sampling can help generate.

Uncertainty sampling is well established and has been proven to work successfully [4]. The downside of this sampling technique is that queried instances in the same batch are often similar and lack *diversity* [5]. This issue can be addressed in different ways, and will be discussed further later.

In order to use uncertainty sampling, one needs to determine a suitable certainty measure. A common uncertainty metric for classification problems is the entropy of the output, where the output is the softmax vector. This vector represents the probability of the data belonging to each class, which can also be interpreted as the models confidence in that the data belongs to each class. The total entropy for the softmax vector is determined by the Shannon entropy equation,

$$E = - \sum_{c=1}^C p_c \log_2(p_c), \quad (2.4)$$

where p_c is the output probability of belonging to class c . To sum over the probabilities over all C classes provides a accumulated measurement for the certainty of the prediction where high uncertainties are penalised. Consider a 3-class classification problem. Our model is uncertain if it outputs the probabilities $[1/3, 1/3, 1/3]$, and certain if it outputs $[1, 0, 0]$. This uncertainty is effectively captured by (2.4).

In the active learning loop, the model iteratively makes queries. These queries can be made in batches, meaning the model selects a number of data points, or it can query one data point at a time. Querying one file per iteration is a simpler task, as diversification doesn't have to be considered. This would also result in a higher resolution, but it requires a lot of computation.

A model is given the option to obtain new data until the performance satisfies some predefined criterion or the budget, or data, runs out. A full active learning loop sampling for a classification model \mathcal{M} is shown in Algorithm 1. This algorithm is quite general, but represents how all active learning frameworks are designed in this thesis.

Algorithm 1 Pool-based Active Learning Loop

Require: $\mathcal{M}, \mathcal{P}_U, \mathcal{P}_L$ ▷ Model, unlabeled data, labeled data
 Initialize \mathcal{M}
 Train \mathcal{M} on \mathcal{P}_L
 Define query strategy Q
while stopping criterion not met **do** ▷ Iterate until the stopping criterion or out of budget
 Select \mathcal{P}_Q from \mathcal{P}_U using Q
 Let Oracle label \mathcal{P}_Q
 $\mathcal{P}_L \leftarrow \mathcal{P}_L \cup \mathcal{P}_Q$ ▷ Add the newly labeled data to the labeled data set
 $\mathcal{P}_U \leftarrow \mathcal{P}_U \setminus \mathcal{P}_Q$ ▷ Remove the newly labeled data from the unlabeled data set
 Train \mathcal{M} on \mathcal{P}_L ▷ Re-train the model using the updated labeled data set
 Evaluate \mathcal{M}
end while

In algorithm 1, the main decision that needs do be made is how to choose the strategy Q . This means deciding how many files are chosen at each iteration, and also how to choose the files. For uncertainty based sampling, Q would be commonly be based on the entropy.

2.2.2 Query Diversification

One of the many challenges when querying data is how to select a good batch, as with uncertainty sampling queries often lack diversity. As the queries in a batch are generally chosen based on the same metric, there is no guarantee that the batch chosen is diverse. We can illustrate this with a simple thought experiment.

Consider the case where we have B identical samples of each data point. Every set of B identical data points would have the same level of model uncertainty, meaning if we select a batch of size B , all queries would be identical. This is obviously not a good property of uncertainty sampling, as the model would benefit from variety in data rather than seeing the same examples multiple times.

We can also imagine a model that has poor performance for a certain class. The active querying strategy will hopefully select data from that class in order to improve. However, it might be superfluous to fill the entire queries of the same class, in this case we are interested in diversifying our batches. Inter-class diversification is also of interest based on the same argument. If you want to teach a child what an apple is, you do not only show the child big green apples, but also small red apples. Diversification can be added to an active learning framework by adding noise to the uncertainty measure [9] or by actively selecting queries that are different based on some similarity measure.

One alternative is to first consider a batch $\bar{\mathcal{P}}_Q$ which is larger than the final batch, $\bar{\mathcal{P}}_Q > \mathcal{P}_Q$. We will sometimes refer to the initial batch the *pre-batch*. \mathcal{P}_Q is selected with the querying strategy Q , for instance some uncertainty based method. The idea is to then select the final queries \mathcal{P}_Q within the

new set $\bar{\mathcal{P}}_Q$ based on some new strategy that enforces diversification in the batch. This approach to active learning with diversification is presented in algorithm 2. An example of a previous study that has used this type of two-step selection is the mismatch-first farthest traversal framework from [5].

Algorithm 2 Pool-based Active Learning Loop with Diversification

Require: $\mathcal{M}, \mathcal{P}_U, \mathcal{P}_L$
 Initialize \mathcal{M}
 Train \mathcal{M} on \mathcal{P}_L
 Initialize query strategy Q
 Initialize diversification strategy \mathcal{S} ▷ Choose a diversification strategy for choosing data to label
while not stopping criterion **do**
 $\bar{\mathcal{P}}_Q$ from \mathcal{P}_U using Q ▷ Select data to diversify from the unlabeled data set using strategy
 \mathcal{P}_Q from $\bar{\mathcal{P}}_Q$ using \mathcal{S} ▷ Select data to label from the subset using div. strategy
 Let Oracle label \mathcal{P}_Q
 $\mathcal{P}_L \leftarrow \mathcal{P}_L \cup \mathcal{P}_Q$ ▷ Add the newly labeled data to the labeled data set
 $\mathcal{P}_U \leftarrow \mathcal{P}_U \setminus \mathcal{P}_Q$ ▷ Remove the newly labeled data from the unlabeled data set
 Train \mathcal{M} on \mathcal{P}_L ▷ Re-train the model using the updated labeled data set
 Evaluate \mathcal{M}
end while

The relative size of $\bar{\mathcal{P}}_Q$ to \mathcal{P}_Q is of interest as there is a trade-off at play. With a bigger $\bar{\mathcal{P}}_Q$ we are more likely to be able to select a diverse batch, but the initial query strategy Q is neglected in favor of diversification. There are multiple ways of selecting a diverse subset of the data points in $\bar{\mathcal{P}}_Q$. We will go over two methods of doing this.

With a good querying strategy Q all data points in $\bar{\mathcal{P}}_Q$ should be of interest for the improvement of the model, but only \mathcal{P}_Q queries can be chosen. One solution is to randomly select from the batch, which adds a bit of stochasticity in the querying, which adds diversity. Another alternative is to use an active second selection strategy. The one used in this thesis is called *Farthest Traversal* [5]. This method can diversify batches by utilizing each data points embedding. The data points that with the largest distance from other data points in the embedding space is chosen. Distance and similarity between two points in the feature space can be described by many different metrics, including *cosine distance/ cosine similarity*. Mathematically this is not a distance metric but rather a way of comparing a similarity between two vectors based on the angle between them in an inner product space, rather than the magnitude in distance between them. The angle is derived by the *cosine similarity* which is the normalised dot product between two arbitrary vectors of the same dimensions.

$$\text{cosine similarity} = \cos \theta = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|} \quad (2.5)$$

The cosine distance is related to the cosine similarity in the following way:

$$\text{cosine distance} = 1 - \text{cosine similarity} = 1 - \cos \theta = 1 - \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|} \quad (2.6)$$

Note that if both vectors are normalised, i.e. $\|\mathbf{x}_1\| = \|\mathbf{x}_2\| = 1$, the equation gets much simpler (2.6). In this thesis the cosine distance will be used, where a high distance is prioritised in the diversification strategy. The data point in $\bar{\mathcal{P}}_Q$ with the farthest distance to its closest data point in \mathcal{P}_L is chosen, labeled and then added to the \mathcal{P}_L . This continues until enough data points are chosen for the query of that iteration. Algorithm 3 shows the complete loop of farthest traversal.

Algorithm 3 Farthest-traversal

Require: $\bar{\mathcal{P}}_Q, \mathcal{P}_L, \mathcal{M}_{embed}$
 $\bar{\mathcal{P}}_{QE} \leftarrow \bar{\mathcal{P}}_Q$ with \mathcal{M}_{embed} ▷ Embed all files in pre-batch
 $\mathcal{P}_{LE} \leftarrow \mathcal{P}_L$ with \mathcal{M}_{embed} ▷ Embed all files in labeled data set
 $\mathcal{P}_Q = \emptyset$ ▷ Empty query
while $|\mathcal{P}_Q| <$ query size **do**
 Min. dist. between $\bar{\mathcal{P}}_{QE}$ and $\mathcal{P}_{LE} \cup \mathcal{P}_Q$ ▷ Calculate minimum distance for all points in $\bar{\mathcal{P}}_{QE}$
 Get p_{QE} , where $p_{QE} \in \bar{\mathcal{P}}_{QE}$ ▷ Extract data point with the largest minimum distance
 $\mathcal{P}_Q \leftarrow \mathcal{P}_Q \cup p_{QE}$ ▷ Add the data point to the query
end while
Return \mathcal{P}_Q ▷ Finally return the query to be labeled

2.2.3 Active Learning for SED

In SED, when using segment based methods, it is reasonable for the active learning framework to query multiple segments. The reason for this restriction is that an annotator can probably not make sense of short segments without any context, thus the oracle is not able to supply a ground truth. In this thesis, entire audio files are considered as data points, and these are queried in full.

The certainty given from the model predictions are for each segment, but as we are interested in querying full audio files, an accumulation method has to be determined as a representation of the uncertainty or entropy in the entire file. There are many possibilities for this metric, and it has mainly been studied in an object detection setting when studying images [8] [7]. In this thesis, multiple strategies, called *aggregation strategies*, based on the individual entropies for each segment will be discussed and evaluated. To present them we establish the following sets. Let \mathbb{E} be a set of the entropies from all segments in a single audio file. \mathbb{E}_e is the set of all the entropies from the segments being **predicted** as events in that file, where $\mathbb{E}_e \subseteq \mathbb{E}$. The following aggregation strategies will be tested as part of uncertainty based querying strategies and are very important for this thesis:

- The *mean entropy* strategy is where the mean of all the entropies in the audio file represent the uncertainty of the file, i.e. $\bar{\mathbb{E}}$. This assures that all uncertainties in the file contributes equally to the overall uncertainty representation of the file. Figure 2.7 shows an example of how mean entropy works for a file containing 5 segments. The green square represents the numerical uncertainty representation the file retains using this method.

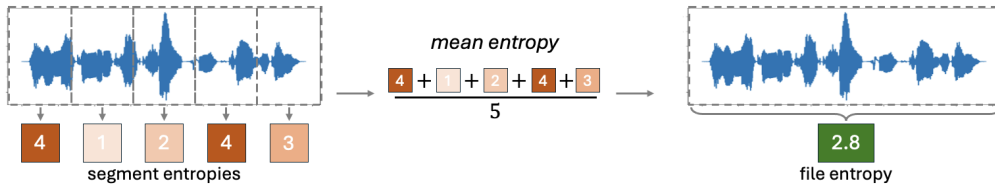


Figure 2.7: **Example: Mean Entropy** - An example of how mean entropy works when accumulating an uncertainty representation for a file using the entropies obtained for the 5 segments of the file. The green square represents the numerical uncertainty representation of the file.

- The *median entropy* strategy is where the median of all the entropies in \mathbb{E} in the audio file represent the uncertainty of the file. This assures that outliers and extreme values does not affect the overall uncertainty representation. Figure 2.8 shows an example of how median entropy works for a file containing 5 segments. The green square represents the numerical uncertainty representation the file retains using this method.

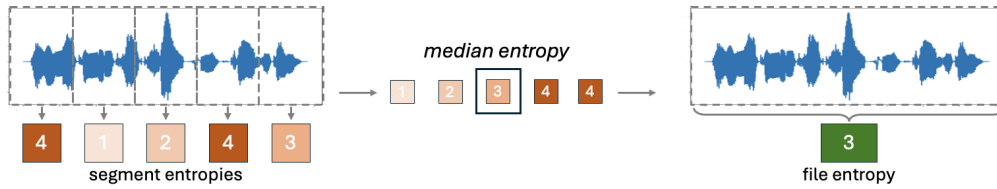


Figure 2.8: **Example: Median Entropy** - An example of how median entropy works when deciding on an uncertainty representation for a file using the entropies obtained for the 5 segments of the file. The green square represents the numerical uncertainty representation of the file.

- The *mean event entropy* strategy is where the mean of all the entropies for the segments that are **predicted** as events in the audio file represent the uncertainty of the file, i.e. \mathbb{E}_e . This assures that only the models predictions of the events are the ones affecting the overall uncertainty representation of the file. Figure 2.9 shows an example of how mean event entropy works for a file containing 5 segments. The blue colored segments are the ones predicted as events by the model. The green square represents the numerical uncertainty representation the file retains using this method.

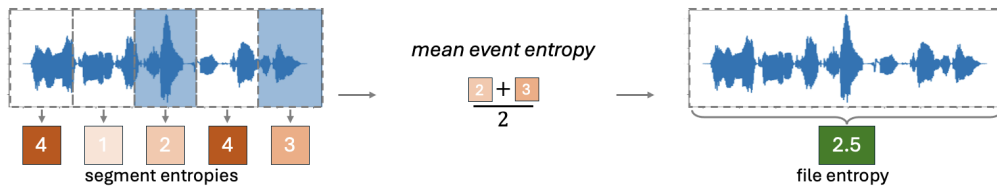


Figure 2.9: **Example: Mean Event Entropy** - An example of how mean event entropy works when accumulating an uncertainty representation for a file using only the entropies that are obtained from segments that are predicted as events, the blue colored segments. The green square represents the numerical uncertainty representation of the file.

- The *top X entropy* strategy is where the average of the X largest entropies in the audio file represent the uncertainty of the file. This assures that only the segments that the model finds the most difficult to classify affects the entropy of the file. Thus this strategy prioritizes files that contain X segments with very high uncertainty. Figure 2.10 shows an example of how top X entropy works for a file containing 5 segments. X is set to 3 in this case, and thus only the 3 largest entropies affect the representation. The green square represents the numerical uncertainty representation the file retains using this method.

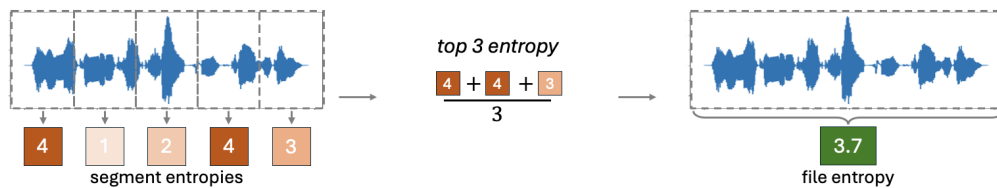


Figure 2.10: **Example: top X entropy** - An example of how top 3 (X) entropy works when accumulating an uncertainty representation for a file using the top 3 entropies obtained from the 5 segments of the file. The green square represents the numerical uncertainty representation of the file.

The aggregation strategies give a single numeric measure of uncertainty per file. The querying strategy based on each aggregation simply takes the $|\mathcal{P}_Q|$ files with the largest entropy.

2.2.4 Performance Metrics

In active learning a big part of the performance lies in the models ability to reach high accuracy fast and one should choose a metric that takes this into consideration while also rewarding the model for its actual prediction performance.

To evaluate the querying strategy in the active learning loop, and its ability to reach high accuracy fast, it is good to have a random query strategy for comparison. This means that instead of choosing queries actively, random selections are made which resembles how a model might receive data when no active learning is used [4]. This strategy is called a *baseline*, or *baseline strategy*. The baseline allows for a more elaborate analysis of using active learning. It is no longer the absolute values of performance that matters, but rather how much the model benefits from active queries. A simple way of evaluating the two is to simply compare different performance measures of these two models at each iteration.

It is also possible to design performance measures that summarise active learning performance. There is no standardised approach for this. One metric used is called *area under the curve (AUC)*. This metric is based on the models performance with regards to some metric, and takes into account how quickly the model learns as this gives affects the area under the performance curve. Figure 2.11 shows two hypothetical accuracy curves for two different strategies and the AUC, \mathcal{A}_1 and \mathcal{A}_2 , for both. The figure also shows the difference in AUC between the two models. This is a metric that is useful to see how large of a difference it is between the models. It is evident that the AUC metric considers and rewards a steep initial learning curve as this increases the total area, which is preferable as it is an important feature of active learning. Although this metric will not be used further in this thesis, it provides an intuitive way to analyze these types of performance graphs.

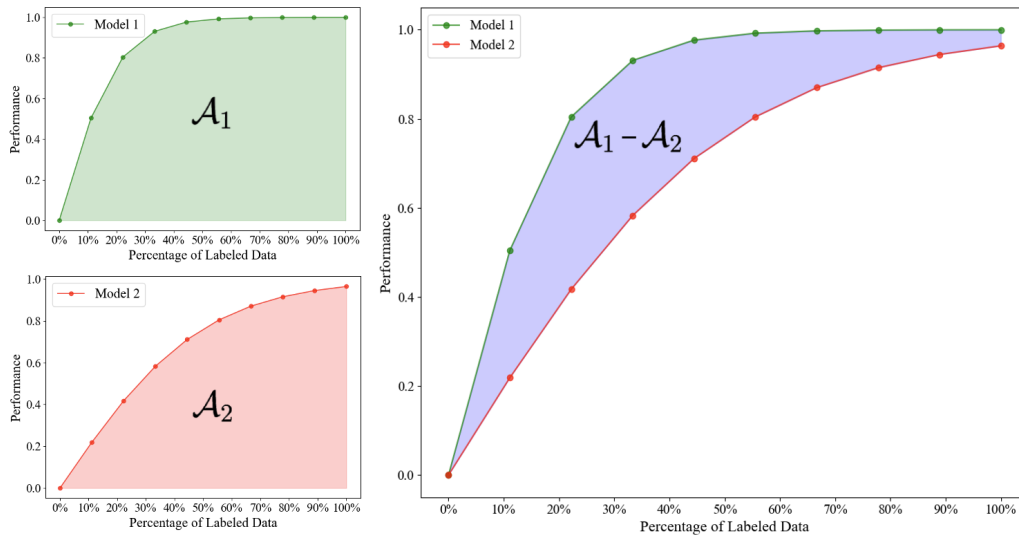


Figure 2.11: **Area Under Curve** - (left) The hypothetical accuracy curves, and their corresponding AUC, \mathcal{A}_1 and \mathcal{A}_2 . (right) The difference in AUC between the two curves.

When evaluating active learning, it is possible to consider differences in how much data is needed to reach a specific performance. One can compare the amount data needed for different models to reach the same, or similar, performance. Figure 2.12 shows an example of this, where the green model seems to reach the same performance as red much earlier. These types of budget reductions should be considered when analysing active learning performance.

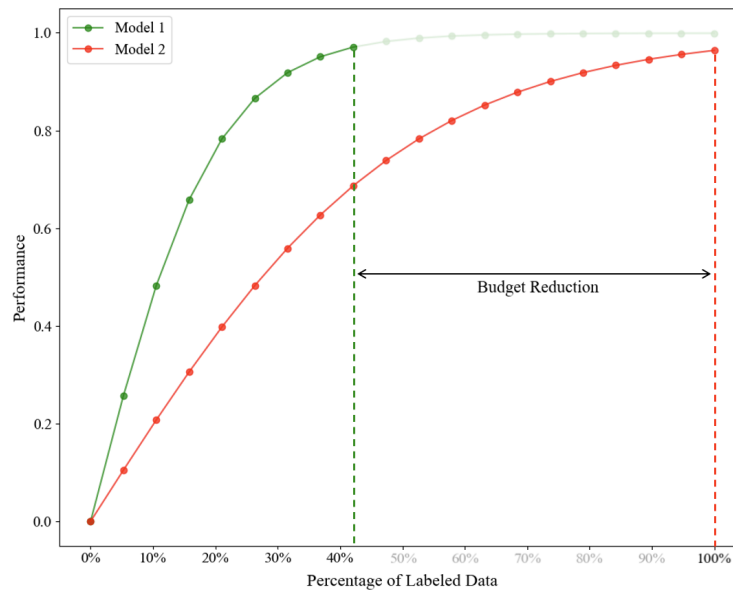


Figure 2.12: **Budget Reduction** - A visualisation of comparing the performance at different iterations between two models. The budget reduction of using the green model and still reach the top performance of the red model is shown.

2.3 Machine Learning Fundamentals

This section is targeted towards those who are inexperienced in the field of machine learning. Anyone with fundamental knowledge about supervised learning and neural networks is encouraged to skip ahead to chapter 3.

In order to consider the specific task of SED, it is important to lay-out some of the fundamentals of machine learning as these play an important part in today's state-of-the-art detection models. This section serves as a short introduction to some of the more fundamental machine learning concepts, with a focus on the ones that are relevant to the field of audio analysis and SED. The goal of this section is to make this thesis more accessible to a broader audience.

What is meant by Machine Learning?

A commonly cited and nowadays famous description of machine learning is given by Mitchell [25]

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ."

Whilst providing an abstract definition, it effectively reduces the concept of machine learning down to three fundamentals: an experience E , a task T and a performance P . What these three components represent depends on the problem at hand, and how we attempt to solve it. The experience usually refers to the information or *data* that is available. The task is the well formulated problem that is being solved. The performance is a preferred *metric* that measures how well the model is at solving the task.

2.3.1 Data Set

A collection of data points is often referred to as a *data set* which we denote $\mathcal{X} \subseteq \mathbb{R}^M$. Each data point $\mathbf{x} \in \mathcal{X}$ itself consists of a set of M *features*, where each feature can be interpreted as a description of an attribute that the data point possesses. **Example:** Consider a class of students, this can be interpreted as a data set where each student is a data point. Each student can be described, and differentiated, by a set of features. These features can be numerical, such as their height or shoe size. The features

can also be categorical, such as hair color or favorite subject in school.

In machine learning we often refer to a *labeled data set*. The data set \mathcal{X} is then accompanied by an additional set \mathcal{Y} consisting of different *labels* \mathbf{y} assigned to each data point \mathbf{x} . These labels are sometimes called *true labels* or *ground truths*. Essentially, the labels are just features, but they are the features that the machine learning model will try to predict, which is why we like to keep them separated. In the context of bioacoustics, the data point \mathbf{x} could be an audio file containing an animal vocalization with the label \mathbf{y} being which species is associated with the sound. Labels can be both categorical or numerical. If the label is categorical, we speak of a classification task, and if it is numerical, a regression task. Going forward, most explanations will be centered around classification tasks as these are of higher importance for this thesis.

Class imbalance

A general problem with all types of classification tasks is the occurrence of *class imbalance*. This is when a vast majority of the data set belongs to a certain class, whilst other classes lack support in the data set. It is easy to imagine that this is almost always the case when presented with bioacoustic data, as there are species and animals more common or louder than others, and thus occur more when the data is gathered. This becomes an even bigger problem when background noise, or silence, is defined as a class, as the vast majority of bioacoustic data might consist of background noise.

Class imbalance can pose a problem in two ways. The first is when *training* the model. If a machine learning model is faced with more examples of a certain class, it could create a bias towards predicting said class. The second problem with class imbalance is when evaluating the model. If one class represents the vast majority of the data set, general performance measures might only reflect the performance of the majority class. This problem can be addressed by considering other performance measures, making sure that different classes are all reflected.

2.3.2 Training

Providing experience to a model by presenting it to data is a part of what is called *training* the model. Depending on the purpose of the model this can be done in different ways. Machine learning methods can generally be divided into two categories, *supervised* and *unsupervised*. Supervised learning is when a model learns to explicitly predict the labels of a data set, by being presented to data. Unsupervised learning refers to trying to find patterns and structure in the data, without knowing what to look for. The difference is explained by the names, where a supervised machine learning model is being supervised by the label \mathbf{y} . In a supervised context, models explicitly predict the label, or *target variable*.

Another useful application of supervised learning is for regression problems, where one tries to predict numerical values, usually by fitting a function that follows the curvature of the given data. One basic example is the problem of linear regression, that is to fit a linear function $f(x) = kx + m$ to some measurements $(x_n, y_n), n \in \{1, 2, \dots, N\}$. A different example of a regression task, that is related to bioacoustics, could be predicting the start or end time of a dog bark in auditory data \mathbf{x} .

The unsupervised approach does not need any explicit label of the data, and focus rather on finding patterns and relations between the data points in the given set. This can be used to group or *cluster* data that is similar.

Classification

Classification problems are often solved using supervised learning. The fact that each data point in the data set has a true label, in the form of a target variable, allows for a straight forward approach to determine how well the model performs as the label can be compared with the prediction. One common performance measure for classification tasks is the *accuracy* of the model, which is simply determined as the ratio between the number of correctly assigned labels and the total number of predictions. This metric is useful and often used for classification problems, where a data point is assigned a discrete class and the labeled data set contains the class corresponding to each data point. The special case

where only two classes occur is called a *binary classification problem*. When multiple classes occur, the task is called *multi-class classification*.

In order to represent a true label in a multi-class classification problem it is common to use a so called *one hot encoding vector*. This representation of the class is what the model sees and tries to predict. The one hot vector is a binary vector that represents each class and usually contains as many elements as there are possible classes. The value on the i :th position determines the data points belonging to the i :th class. Where 1 usually refers to the data point belonging and 0 meaning that it does not. This representation will be important later, when we provide a more detailed explanation of how classification models are constructed and trained.

If we have three classes, *baby*, *dog* and *meerkat*, each class has a unique representation in the one hot representation. For example, the representation could be

$$\begin{aligned} \text{"baby"} &\rightarrow [1, 0, 0] \\ \text{"dog"} &\rightarrow [0, 1, 0] \\ \text{"meerkat"} &\rightarrow [0, 0, 1] \end{aligned} \tag{2.7}$$

Loss function

A fundamental part of training is the ability to determine if improvements are being made, i.e. if the training is successful. In machine learning this is achieved through the so called *loss function* $\mathcal{L}(\theta)$. The loss function is dependent on the model parameters θ and is a type of performance measure, usually related to some average error between the model prediction of a datapoint $f(\mathbf{x}_i; \theta)$ and the ground truth \mathbf{y}_i .

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{y}_n, f(\mathbf{x}_n; \theta)) \tag{2.8}$$

In (2.8), ℓ is the *loss function* for a single prediction. Accuracy has been introduced as a metric of how well models performs when faced with a classification problem. This is a discrete measurement, either the model is correct or not. This means that accuracy isn't differentiable which is necessary in order to optimise the loss function with standard methods. Let's consider an example to illustrate this principal.

Imagine shooting 3-pointers in basketball. If you continuously hit the rim after each throw, you know that you are close to your goal and just need small adjustments. Compare this to throwing the ball with your eyes closed, meaning you don't see how close you are unless you hear the ball going through the net. If you don't see where the ball lands, it is hard to know what adjustments to make. Whilst your accuracy is the most interesting metric of how good of a basketball player you are, it is not the most crucial information for you to improve. This is the difference between a general performance metric and a good loss function. When evaluating a model, accuracy might be a good sign of performance, but it is not the feedback it needs to train and improve.

There are multiple different loss functions, applicable for different purposes. Consider a linear regression problem. Let's assume we have a data set $\mathcal{X} \in \mathbb{R}^2$ and wish to fit a straight line. The distance between the value \mathbf{y}_n of the data point \mathbf{x}_n and its corresponding value $\hat{\mathbf{y}}_n$ on the regression line determines how well the line fits for point n , see figure 2.13. The mean of all the squared *errors* gives us a good loss function for this problem. This is called the *Mean Squared Error*:

$$MSE = \ell(\mathbf{y}_n, \hat{\mathbf{y}}_n) = (\mathbf{y}_n - \hat{\mathbf{y}}_n)^2 \tag{2.9}$$

Training the model can now be reduced to optimising the loss function. This will be covered in its own section, but it is important to note that it is often times helpful if the loss function is differentiable. Many optimisation methods use the derivative to update the model parameters in a favorable way.

Having described the use and purpose of a loss function it is time to introduce the *categorical cross-entropy loss*, also called the *softmax loss*, as it is the loss function used in this thesis. It is customary

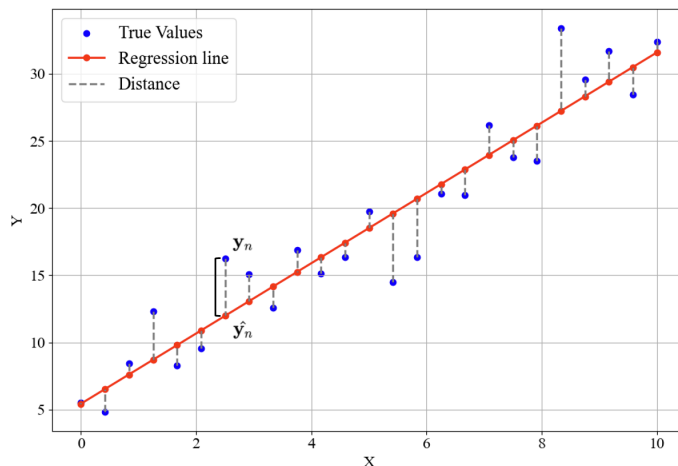


Figure 2.13: **Mean Squared Error** - visualisation of MSE. The difference, $y_n - \hat{y}_n$, for the true- and predicted value of point \mathbf{x}_n is marked with a solid black line. The regression line shows the line of which the mean of the square of all the distances (black dashed lines) is as small as possible, i.e. the line which reflects the underlying data as good as possible.

to use this loss when faced with a multi-class classification problem and combines the *softmax output activation function* and the *cross-entropy loss function*. The purpose of an activation function will be saved for later. What is important now is to know the output of the softmax function, i.e. what it returns. It returns a probability distribution of the data point belonging to each class in form of a vector. In other words a vector where each element represents the probability of belonging to a class. The sum of the vector should be one, as the total probability is one.

In order to calculate the categorical cross-entropy loss, the output vector is compared with the true label in its one hot encoding vector form. Naturally, if the prediction is perfect, this will be identical to one hot representation of the ground truth. The cross-entropy loss function compares the two vectors with the following equation:

$$\ell(\mathbf{y}_n, \hat{\mathbf{y}}_n) = CE_n = - \sum_{c=1}^C y_n^c \log_2(\hat{y}_n^c) \quad (2.10)$$

\hat{y}_n^c is the predicted probability of class c for data point n , whereas y_c is the one hot encoded ground truth for data point n and class c . Taking the mean of all these errors for all predicted data points n gives us the full cross-entropy loss function [16].

Performance Metrics

In order to compare and evaluate models after training, a reasonable performance metric is vital. In order to develop this theory, we will consider a *positive class* (yes) and a *negative class* (no). In this thesis, when different animal vocalizations are studied, all animal classes are assigned to the positive class whereas background noise is assigned to the negative class.

If a model correctly identifies a positive data point, this is referred to as a *True Positive*, or *TP*. Similarly, a *True Negative* *TN* is a correct classification of the negative class. Conversely, *False Positives* (*FP*) or *False Negatives* (*FN*) relate to incorrect classifications of the positive and negative classes, respectively. These four metrics can be used to construct other types of metrics that provide different perspectives of the models performance. A graphical interpretation often called a *confusion matrix* is shown in figure 2.14.

		True Label	
		Yes	No
Predicted Label	Yes	TP	FP
	No	FN	TN

Figure 2.14: **Confusion matrix** - A general confusion matrix for a binary classification problem. TP (True Positive) shows how many correctly predicted labels of class "yes". TN (True Negative) shows how many correctly predicted labels of class "no". FP (False Positive) shows how many wrongfully predicted labels of class "yes". FN (False Negative) shows how many wrongfully predicted labels of class "no". A large amount on the diagonal of the matrix is desired.

One previously mentioned metric is *accuracy*, which is the ratio of correct predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.11)$$

This metric is commonly used, but is not ideal when faced with class imbalance. Imagine if 90% of data points belong to the negative class. The model can then predict all labels being negatives and the accuracy would turn out to be relatively good. This type of class imbalance is common in real life data and is a challenge faced in this thesis, which makes accuracy a less interesting metric. A better metric in these types of situations is *precision*, and is calculated in the following way:

$$Precision = \frac{TP}{TP + FP} \quad (2.12)$$

It can be interpreted as how good the model actually predicts the positive class, as all predictions that involve the negative class is no longer taken into account. This makes precision suitable for imbalanced problems, where some classes might be more important to evaluate than others. An additional metric that is suitable in these situations is *recall*, and is calculated in the following way:

$$Recall = \frac{TP}{TP + FN} \quad (2.13)$$

It looks similar to precision, but can be interpreted as how many instances of the positive class did the model manage to predict correctly. As the property of both these metrics are important there is a third metric that takes the harmonic mean between the two. This is the so called *F1-score*, and is calculated in the following way:

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (2.14)$$

It is important to not that the division into a *positive* and *negative* class is not only useful when dealing with a binary yes/no classification problem. It can easily be adopted to a multi class scenario in two ways. The first alternative is to look at one class at a time, which will represent the positive class. The other is to assign a selection of classes to the positive class and the rest to the negative class.

2.3.3 Optimisation

We have discussed training as a way of making a model better in terms of presenting it to data and minimizing a loss function. We haven't discussed the optimisation theory that goes into doing this successfully. How do we find the correct model-parameters θ for our machine learning model? The fundamental part of the optimisation is to minimise the loss function $\mathcal{L}(\theta)$, i.e. find the arguments that

provides the minimal value of error and thus the optimal solution. It is favorable if the optimisation is not to computational expensive. For further explanations we assume our loss function to be differentiable on a closed interval. The *Extreme value theorem* then holds as the function is continuous and this assures that a minimum value exists. The optimisation algorithms in this section will be visualised in two dimensions but are all scalable for larger dimensions. Generally, optimisation algorithms relies on an initial guess θ_0 . The initialization is often random, which means these algorithms can differ in time and computation, and a number of trials might be necessary. Evaluating the loss function at each positions allows for an understanding of how the weights should be updated to improve performance. As the function is differentiable the gradient of the function can be determined in each point, and thus the direction of which a lower value can be reached. This approach is called *Gradient Descent* (GD) where every new step at each time t is being updated by the gradient of the loss function at the current position:

$$\theta_{t+1} = \theta_t - \gamma \nabla \mathcal{L}(\theta_t) \quad (2.15)$$

The subtraction sign is due to the fact that a gradients points in the direction of which the function has its highest increase rate. The *gamma* is called the *learning rate*, or *step size* and determines the magnitude of the step taken at each iteration. This is a so called *hyperparameter* as it has to be determined in advance and can be tweaked for better performance. The learning rate is usually in the range of $(0, 1]$ and if it is too small the minimum can be hard to reach as too many iterations might be needed. If too small, the optimisation might converge to a *local* minimum i.e the minimum in some interval. This means that the method fails to find the *global* minimum which is the minimum over all θ for which the function is defined. If the learning rate is too high then the minimum can easily be missed as the method might fail to converge. Figure 2.15 shows how a GD search gets stuck in a local minimum and thus does not converge to the optimal solution of the problem.

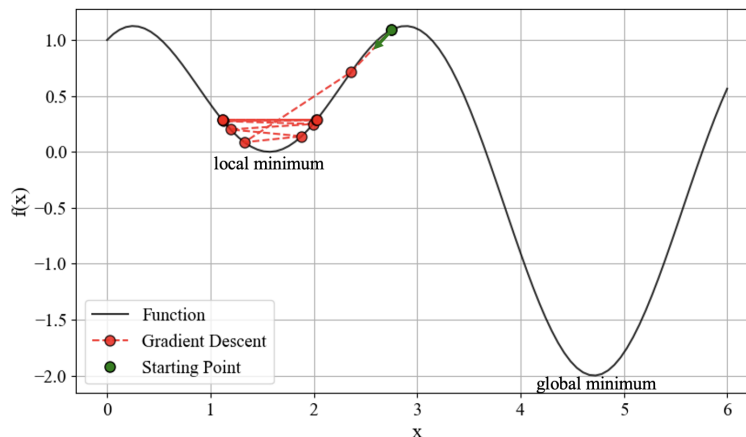


Figure 2.15: **Gradient Descent - Global- and local minimum** - The phenomenon of Gradient Descent (GD) getting stuck in a *local* minimum. This prevents convergence to the global minimum.

One way to avoid converging to a local minima is to repeat the optimisation multiple times with different initial guesses and then hopefully find the global minima of the problem. This becomes very computational expensive as a lot of gradients have to be calculated, especially if the data set is large. To solve this an approximation of the gradient can be used instead. In *Stochastic Gradient Descent* (SGD) a random *mini-batch* of data points are chosen to represents the whole data set, and the gradient of the loss of that data point is used as the update term instead.

$$\theta_{t+1} = \theta_t - \gamma \nabla \mathcal{L}_i(\theta_t) \quad (2.16)$$

Here $\mathcal{L}_i(\theta)$ relates to the loss for batch number i . The stochastic part of SGD is the division in to mini-batches. This does not only improve the computational time but also provides stochasticity in the stepping at each iteration, as a mini-batch rarely represents a whole data set perfectly. The stochasticity tends to help with the problem of getting stuck in local minima. As the search is inexact it usually converges toward a less accurate minima after more iterations as a trade-off for its computational

efficiency. It is possible to have an *adaptive learning rate* which allows for the learning rate to change for each iteration. This has the potential of combining quick convergence and precise results. In 2014 the Adaptive Moment Estimation, or ADAM, was introduced. It uses the momentum at each step by taking the previous directions into account. These are weighted with an exponentially decreasing factor which rewards more previous directions over old ones. The new accumulated direction is then also multiplied with a step size for easy magnitude adjustments before updating the step. Algorithm 4 shows the full loop [26]. ADAM includes more parameters to initialize, but these usually need less tuning than parameters in other optimisation methods.

Algorithm 4 ADAM Optimisation Algorithm

Require: Learning rate γ , exponential decay rates β_1, β_2 , and a small constant ϵ .

$m_0 \leftarrow 0$

$v_0 \leftarrow 0$

Initialize θ_0

▷ Initialize guess for parameters θ_0

$t \leftarrow 0$

while not stopping condition **do**

▷ Loop until minimum reached or max number of iteration

$t \leftarrow t + 1$

▷ Take a step

Compute gradient $g_t \leftarrow \nabla f(\theta_{t-1})$

Update biased first moment estimate: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$

Update biased second moment estimate: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$

Compute bias-corrected first moment estimate: $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

Compute bias-corrected second moment estimate: $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

Update parameters: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

end while

Final parameters θ_t

2.3.4 Neural Networks

So far, the theory has revolved about how data might be used in machine learning and how to optimise models. Little has been said about what a machine learning model might look like. In this section we introduce the *neural network*, a model that comes in many shapes and forms. The neural network is of importance in this thesis as more advanced versions of it are used to extract features from auditory data, and a simple network is used to classify animal vocalizations.

Neural networks are universal function approximators, meaning they can approximate any function. As most things can be described as functions, neural networks are a very powerful tool to have in your machine learning toolbox. Whilst it is not feasible to give a comprehensive explanation of neural networks and their parameters, we attempt to give some insight to the topic here.

A neural network consists of nodes ordered into layers with edges connected to other nodes. Figure 2.16 is a visual representation of this structure. The nodes in the first layer represent the input data \mathbf{x} . A neural network essentially takes a weighted sum of the nodes in the first layer, then applies some (typically non-linear) transformation and then repeats this process in the next layer. Each edge has a weight, ω , which is used in the mentioned sum. This means that a node h in the next layer is assigned a value by taking the weighted sum a of the nodes that point to h . This weighted sum a is fed through a function, called the *activation function*, which gives the node its final value h . The nodes are updated in a *feed-forward* way. In figure 2.16 this means that information is passed from left to right. The weights are selected by optimising some loss function. In doing so, the neural network approximates the function that maps the data \mathbf{x} to the label \mathbf{y} .

A bias, b is usually added to each node, when calculating the weighted sum a . The bias is also a weight that is added to the input to the activation function. For a simple neural network that represents a linear function, we can interpret the weights ω as the slope of the function, and the bias as the intersect. Note that the bias is not depicted in figure 2.16.

Activation functions are often what determines what types of function the network can approximate.

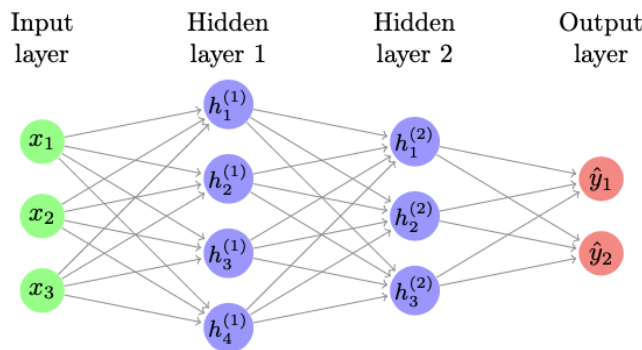


Figure 2.16: **Small Neural Network** - An example of a small neural network containing two hidden layers. The network is fully connected and has 3 input nodes, and 2 output nodes.

If the activation functions are linear, $\phi(a) = a$, the network can only capture linear relationships in the data. Other possible choices of ϕ include the Rectified Linear Function $\phi(\mathbf{a}) = \max(0, a)$, or the tan-hyperbolicus $\phi(a) = \tanh(a)$, both of which are popular options.

As an example we consider a node in the second layer of the neural network in figure 2.16: $h_1^{(2)}$. Its value will be affected by the weights assigned to the edges pointing to the node. There are $K = 3$ such edges, the weights of which we denote ω_k . We denote the bias that affects our node as b (not displayed in the network in figure 2.16). Finally, given an activation function ϕ_h assigned to our node we can calculate the value of the node as,

$$h_1^{(2)} = \phi(a_1^2) = \phi\left(\sum_{k=1}^K \omega_k h_k^{(1)} + b^{(1)}\right) \quad (2.17)$$

In order to extend this to a full network, the notation of weights and biases would have to change to account for different layers and nodes. This notation would look more intimidating, but it would be the same core idea at play. [16]

Output nodes

The output of the neural network $\hat{\mathbf{y}}$ is represented by a set of nodes. In a regression task, when trying to predict a real-number, only one node is needed, usually with a linear activation function. The output node(-s) represent an output vector, that can be compared with the ground truth for that data pattern.

The loss function relies on a comparison between the model output and its true label. This means that the structure of these two has to be the same. By carefully choosing the activation function for the output nodes, one can make sure that the output has the properties necessary in order to be used in the loss function. For the sake of this thesis, only the one used will be introduced. Keep in mind, there are many more. The *softmax activation function* returns an output in form of a vector, \mathbf{y} . The vector has two crucial properties:

- The sum of all elements in the vector equals to 1, $\sum_{c=1}^C y_c = 1$.
- Each element in the vector is constrained to the range between 0 and 1 (exclusively), $y_c \in (0, 1)$

Both these are fundamental properties for the output vector to be the form of a probability distribution. The conversion to probabilities are done using the following equation for each element:

$$y_c = \frac{e^{a_c}}{\sum_{i=1}^C e^{a_i}}, \quad (2.18)$$

where a is the argument provided from the previous layer in the neural network [16].

Chapter 3

Data Generation

A short chapter with an overview of the data used in this thesis. In 3.1, the data generation is explained. This section is followed by an exploration of the generated data in section 3.2. The chapter ends of with a visualisation of the distribution of the data.

3.1 Method

3.1.1 Generating Soundscapes

The data used to study active learning for the SED model is synthetic and generated by mixing foreground sounds with background audio. The foreground audio consist of the events that are supposed to be detected by the model. These audio files are shorter and contain sounds belonging to one out of three classes: **meerkat**, **dog** and (human)-**baby**. The audio used as background noise are recordings from a park, and will be refereed to as noise. The noise has some variability, as it is possible to hear lawn-mowers, birds, foot-steps etc. in these recordings. All data points generated from the background sources are 10 second 44.1 kHz **.wav** files. Each file is mixed with different number of events and paired with a ground truth, specifying a start-time, end-time and class for each event that occurs.

Each data set will be referred to as \mathcal{D}_r^{SNR} . SNR specifies the signal to noise ratio used when mixing the noise with foreground signals, see equation (2.2). The parameter r specifies the ratio between the number of files with events and the total number of files. This means that $1 - r$ is the ratio of files without events, i.e just noise. In this project, $\mathcal{D}_{0.2}^0$ is the primary data set used when studying active learning for SED, but other SNR and r values are tested to study the impact of these two parameters. The size of a data sets is defined by the number of **.wav** files it contains, where $|\mathcal{D}_r^{SNR}| = 2500$ is used in all experiments in this thesis.

There are some stochastic aspects of the data generation. The number of events present in a file, the class assigned to each event and the temporal location of the events in the file are all chosen from a uniform distribution $U(a, b)$. The length of the events are determined by the duration of the foreground files themselves. The classes might have different distributions in terms of length. Remember that only a fraction r of the files in \mathcal{D}_r^{SNR} will contain events. The number of events that each file with events will have is drawn from $U(1, 3)$. Once the number of events is determined, the type of event is chosen uniformly over all three classes. Finally the temporal placement of events is selected by randomly drawing a start time (in seconds) uniformly over an interval, $t_{start} \sim U(0.5, 8)$. The interval facilitates the annotation process for the oracle. If the placement leads to an overlap between two events, t_{start} is drawn once more. The maximum duration of a single event is set to 7 seconds to make room for other events. The data generation explained above is summarised in table 3.1.

Table 3.1: **Summary of Data Generation** - Summary of the data generation process which determines the properties of each data set \mathcal{D}_r^{SNR} . N_e is the stochastic variable determining the number of events in a file, given that it will have events. t_{start} is the start time within a 10s file drawn for each event in the data set.

Parameter	Value
Data set size	$ \mathcal{D}_r^{SNR} $
Ratio of files with events	r (data set dependent)
Signal-to-noise ratio	SNR (data set dependent)
Audio file length [s]	10.0
Maximum event length [s]	7.0
Distribution of N_e	$U(1, 3)$
Distribution of t_{start}	$U(0.5, 8)$
$P(\text{event is baby})$	$P_b = 1/3$
$P(\text{event is dog})$	$P_d = 1/3$
$P(\text{event is meerkat})$	$P_m = 1/3$

3.1.2 Evaluation Data Set

For each data set \mathcal{D}_r^{SNR} , a different data set $\bar{\mathcal{D}}_r^{SNR}$ can be generated. The reason for this is to evaluate how the developed active learning frameworks on \mathcal{D}_r^{SNR} adapt to a slightly different domain, containing unseen events and noise from new audio recordings.

Each evaluation data set $\bar{\mathcal{D}}$ is generated using the same parameters as for \mathcal{D} , see table 3.1. The main difference between the two sets is that \mathcal{D} and $\bar{\mathcal{D}}$ are generated using different recordings, both for the foreground- and the background audio. For instance, when a meerkat event is placed in a .wav file in set \mathcal{D} , the meerkat recording is randomly chosen from a set of meerkat recordings \mathcal{S}_m . When placing a meerkat event in a file in set $\bar{\mathcal{D}}$ the recording is chosen from a different set of meerkat recordings $\bar{\mathcal{S}}_m$, where $\mathcal{S}_m \cap \bar{\mathcal{S}}_m = \emptyset$. The same goes for placing noise, and any other event.

All data sets \mathcal{D}_r^{SNR} used for the studying of active learning are separated, where one part of the data set (80%) is available for training and the other part (20%) is used as a validation set for evaluating model performance. The method for this will be covered in more detail when the methodology to study active learning is explained. The $\bar{\mathcal{D}}_r^{SNR}$ data set could then be used in the same way to see if the framework is able to generalise to a slightly different domain.

3.2 Results from Data Generation

3.2.1 Exploring $\mathcal{D}_{0.2}^0$

As previously mentioned, the $\mathcal{D}_{0.2}^0$ is the most commonly used in this thesis. The event length distribution for each class from the generated data set $\mathcal{D}_{0.2}^0$ is shown in figure 3.1. It is clear that the baby vocalizations are the longest, and meerkat vocalizations are the shortest out of the three classes. Highest variance is observed for baby events, whereas meerkat events have the lowest variance. These properties are also seen in table 3.2 which summarises the distribution of event lengths E_L . The data set data consists of 25 000s of audio. This is made up by 302.7s of baby vocalizations (1.2%), 110s of dog barks (0.4%), 59s of meerkat sounds (0.2%) and 24 529s of noise.

Table 3.2: **Distribution of Event Lengths (E_L) for $\mathcal{D}_{0.2}^0$** - $\mathbb{D}(E_L)$ is the standard deviation of the event lengths. "Me" represents the meerkat class. It is clear that the average length events is longest for baby and shortest for meerkat. The same goes for the standard deviation.

Class	Number of Events	Average E_L	Std(E_L)	max E_L	min E_L
Baby	302	1.00	0.81	7.00	0.08
Dog	341	0.32	0.24	1.54	0.07
Me	339	0.17	0.054	0.29	0.10

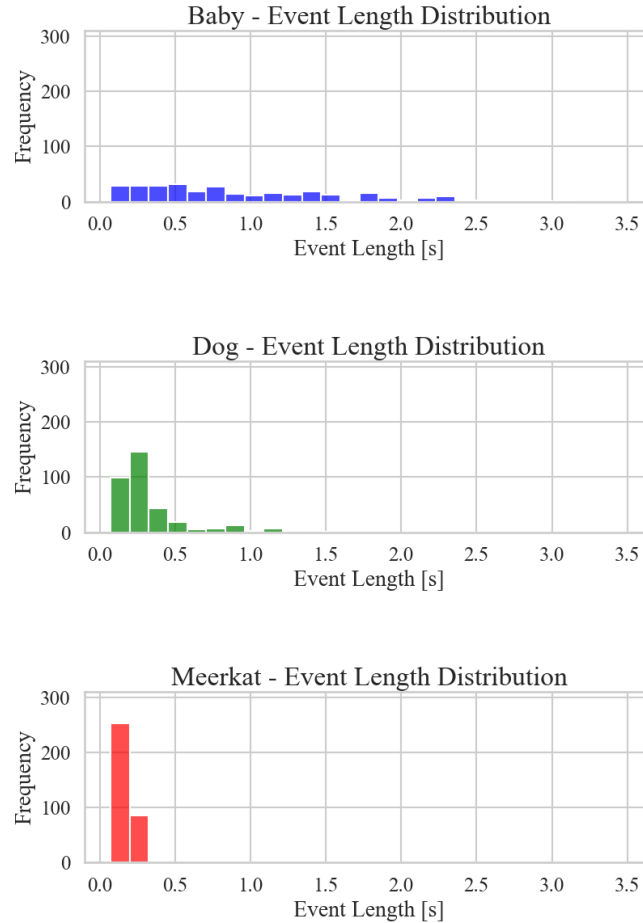


Figure 3.1: **Event Length Distributions** - This plot shows the content of the $\mathcal{D}_{0.2}^0$ data set, in terms of the temporal distribution of each class. As the baby events had one outlier that was 7 seconds long, this event is not displayed in order to better see the distribution of the other events.

3.2.2 Exploring $\bar{\mathcal{D}}_{0.2}^0$

In this section, the same exploration is done for $\bar{\mathcal{D}}_{0.2}^0$ as previously done for $\mathcal{D}_{0.2}^0$. It is important to know how similar $\bar{\mathcal{D}}_{0.2}^0$ and $\mathcal{D}_{0.2}^0$ are, as this sets the context and expectations for the final testing on $\bar{\mathcal{D}}_{0.2}^0$. Differences in the two data sets can help explain potential differences in results.

In table 3.3 the distribution of events is summarised. Most of the class dependent traits seen for $\mathcal{D}_{0.2}^0$ can be observed here, with slightly different values. The total event time is longer in $\bar{\mathcal{D}}_{0.2}^0$. Baby events are on average longer, and dogs on average shorter in $\bar{\mathcal{D}}_{0.2}^0$ compared to $\mathcal{D}_{0.2}^0$. The distribution of event lengths is also shown in figure 3.2. The data set comprises a total of 25 000 s of audio. Among these, there are 352.04 s of baby vocalizations (1.41%) of the data set, 82.61 s of dog barks (0.33%), and 62.20 s of meerkat sounds (0.25%). The majority of the data set, 24 503 s, consists of noise, representing 98% of the total data. In $\mathcal{D}_{0.2}^0$, the events total 471 s, whereas the events in this data set account for 497 s which is a 5% increase in events.

Table 3.3: **Distribution of Event Lengths (E_L)** for $\bar{\mathcal{D}}_{0.2}^0$ - $\mathbb{D}(E_L)$ is the standard deviation of the event lengths. "Me" represents the meerkat class. In general, the distributions look similar in comparison with $\mathcal{D}_{0.2}^0$. The main difference is that the dog events are slightly shorter, and baby events slightly longer.

Class	Number of Events	Average E_L	Std(E_L)	max E_L	min E_L
Baby	300	1.17	0.85	3.61	0.17
Dog	313	0.26	0.18	1.04	0.12
Me	373	0.17	0.05	0.28	0.10

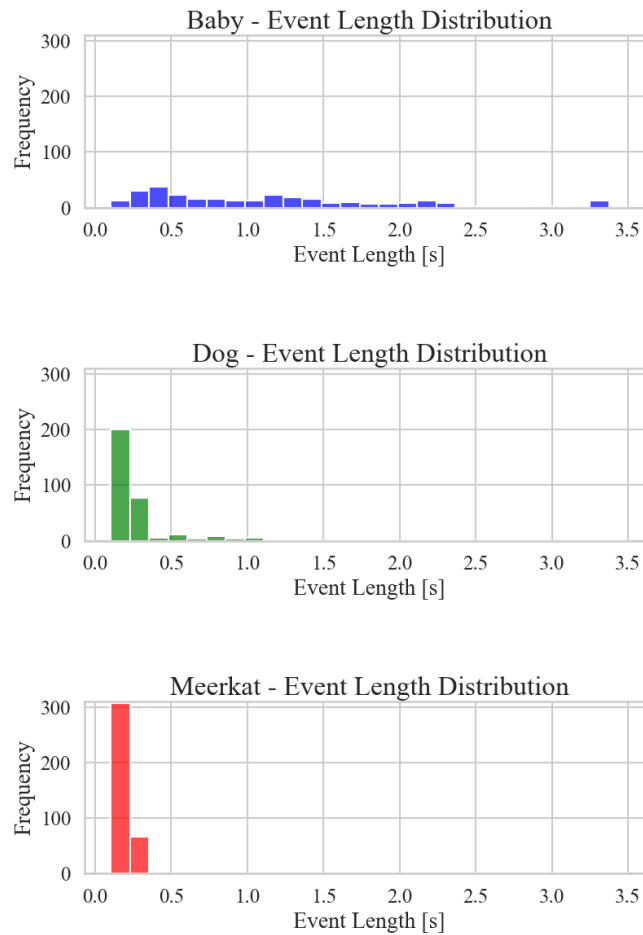


Figure 3.2: **Event Length Distributions** - This plot shows the content of the $\bar{\mathcal{D}}_{0.2}^0$ data set, in terms of the temporal distribution of each class.

Chapter 4

SED Model

This chapter presents the SED model used in this thesis. Section 4.1 gives summary of the entire model along with a visualisation of the full SED pipeline. Following this, section 4.2 breaks down each part of the model. Finally section 4.3 presents some examples that aim to show how the model works.

4.1 Summary of SED Model

The SED model used to study active learning is built using a segment based approach. Each 10 second file in a data set \mathcal{D} is split into short segments and fed through a pre-trained feature extractor called YAMNet (see section 2.1.3, *Advanced Network Architectures*). YAMNet outputs embeddings which are low dimensional representations of the input data. The embedding corresponding to each segment are assigned their ground truth class: noise, baby, dog or meerkat. This is the target variable to predict based on the embeddings. Note that noise is defined as its own class.

A small neural network is built to classify each segment based on the embeddings. This is referred to as the classifier-head, which is trained to predict the class for every segment in every file. The classifier-head has input dimension 1024, corresponding to the YAMNet output embedding dimension, and 4 output nodes, one for each class. Once the classifier is trained, it outputs probabilities for each of the classes, for each time-segment in the files. Figure 4.1 shows the entire pipeline for the class prediction of a single segment.

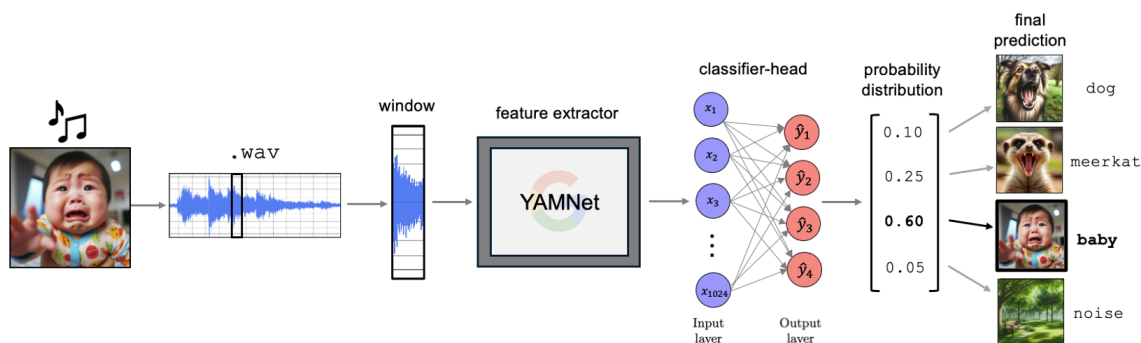


Figure 4.1: **SED Prediction Pipeline** - The pipeline from audio source, in this case a baby, to a final prediction of a single segment using the architecture of the model in this thesis.

For SED, the desired output is a start- and end time for events, meaning we need to process and merge segment predictions into this format through post-processing. This is done by first filtering the

raw output prediction from the classifier-head with a median filter of size 3, which applies the median operator over the time dimension in the classification input. This essentially means that the prediction for a segment is affected by the probability outputs for adjacent segments. After filtering, the adjacent segments assigned the same class are merged into events, which makes up the final predictions of the SED model.

In the sections that follow, more detailed explanations of all the fundamental parts of the SED model is provided.

4.2 Method

4.2.1 Pre-processing

The SED pipeline starts with loading the generated `.wav` files and pre-processing. It is preferable if the file lengths are a multiple of 0.96 seconds as that is the input dimension of YAMNet. For this reason, when loading the files, zeros are added at the end to make the length 10.56 seconds. The sample rate required by YAMNet is 16 kHz (15360 samples). Since the data set contains 44.1 kHz audio files, we can fit more information into YAMNet by not resampling the data to 16 kHz. This is accomplished by using shorter time segments than 0.96 s with the native sample rate of the files. Shorter time segments do not only allow more information to be retained in the embedding, but also allows for a finer resolution for the SED.

In this project, a segment size of $\frac{1}{8} \cdot 0.96 = 0.12$ seconds is used, and each segment overlaps 50% with the next, which allows for even better resolution. As previously mentioned, YAMNet accepts 15360 samples. Zero-padding is used to adjust for this dimensionality, that is making the 0.12 second segment recorded at 44.1 kHz fit into YAMNet. In summary, the file is cut into slices of 0.12 seconds (44.1 kHz), and zeros are added at the beginning and end so that each slice fulfills the requirements from YAMNet. Each file in the data set contains 175 segments after zeros are added to the end. This means that YAMNet outputs a time-series of 175 embeddings.

The goal of the SED is now to classify each small segment, and then combine the segment predictions to intervals to obtain events, characterized by the events start time, end time and class. In order to build a classifier for each segment, they must be paired with a corresponding ground truth. As the raw format of the ground truth is the class, start- and end time, this also needs to be processed in a segment-like manner. Each segment in the file has its own start- and end time, so it is easy to check if a segment overlaps with the interval where a ground truth event is present. However a decision has to be made about *how much* a segment must overlap with a true event in order for it to be assigned the label of that event. If a segment only overlaps with 0.01 s of an event, it may not be reasonable to assign that segment with the event label, see figure 2.2. In this project, a threshold of 50% is set, meaning that a segment must contain more than 50% of an event in order to be assigned the label of that event. This option, along with the segment overlap of 50%, best captures the temporal position of events in the ground truth data. Note that this means that events of duration 0.06 seconds would be completely discarded, but this is not a problem in our case as no such events exists, see table 3.2. The reason behind the 0.12 second segments is to assure that no true events would be discarded by the pre-processing. The class assigned to each segment is represented as a one-hot-encoded vector, which is useful when training the classifier.

4.2.2 Classification

After pre-processing, YAMNet is used to embed all segments. The purpose of this is to reduce dimensionality and extract salient features from the segments to classify them. A classifier-head in the form of a small neural network is used for classification of each segment. In our case, the classifier-head is fully connected and has no hidden layers, i.e it connects all input nodes with all output nodes directly. The input layer consists of 1024 nodes corresponding to each feature in the YAMNet embedding. There are four output nodes, one for each class, with softmax activation functions. This means their value can be interpreted as the models confidence that the input segment embedding belongs to each class.

The final prediction of each segment, corresponds to the node with the highest softmax output.

The classifier-head is the part of the SED pipeline that requires training in a supervised manner. Thus, it is in the training of the classifier-head where active learning is applicable. The training parameters seen in table 4.1 are the ones chosen for our model, as these gave good validation performance when training the model. These hyperparameters are used when training each model at every active learning iteration, this to draw valid conclusions and comparisons.

Table 4.1: **Classifier-head Hyperparameters** - These are the predetermined hyperparameters used to train each model when studying active learning.

Parameter	Value
Input dimension	1024
Output dimension	4
Loss function \mathcal{L}	Categorical Cross-Entropy
Optimisation method	ADAM
Learning rate	0.01
Epochs	50

4.2.3 Post-processing

The raw output from the classifier-head for each segment needs to be processed into event labels consisting of class, start- and end time. This begins with filtering the results, followed by merging predictions into continuous intervals.

A median filter of kernel size 3 is used to remove model noise, see section 2.1.1. The median operator is not linear, meaning we cannot describe the filtering mechanism using traditional convolutions. However, we can think about it much like a convolution. Consider the median operator $m_3(\cdot)$ which returns the median of its input, which is a vector of length 3. For the 175 segments we output 4 class probabilities, meaning the raw output from the model, for each file, is $\mathbb{R}^{4 \times 175}$. The median operator is then applied along the temporal dimension, independently for each of the 4 class predictions. The first and last segments in the file keep their raw output, as these segments are missing one of their neighbouring segments.

As an example, consider that three consecutive segments are assigned labels "meerkat, baby, meerkat" by the classifier-head. After filtering, the baby prediction in this example will likely be convinced by it's neighbours that it should be labeled as a meerkat. This is one of the advantages of the filtering process; that single predictions can never break up entire events. This is an important feature to have in the post-processing, as the segments overlap.

Following the filtering process, any single segment events that remain are discarded. This is reasonable as we have previously established that no event can only be covered by one single segment. Segments are merged into an interval, described by its start time, end time and class, for all consecutive segments that belong to the same class. The processing is now complete, meaning the output from this stage is the final model output for the SED.

4.3 Results

4.3.1 Visualisation of Model Output

In figures 4.2 and 4.3 the final SED predictions are visualised against the ground truth. The purpose of these figures is mainly to see how segments are classified and then processed into events. In this case the model was trained on a small data set of 100 files, with $SNR = 3$ dB, for visualisation purposes.

In the bottom subplots of both figures, the classifier-head output, before post-processing, is visualised. In figure 4.2, it is clear that the predictions are good in terms of placement in the time domain, but it struggles with some of the segment labels. However, after filtering the predictions are nearly perfect, as seen in the top subplot. In figure 4.3 we see a similar example, but in this case we see the impact of the filters removal of single segment predictions, where it manages to remove false positives.

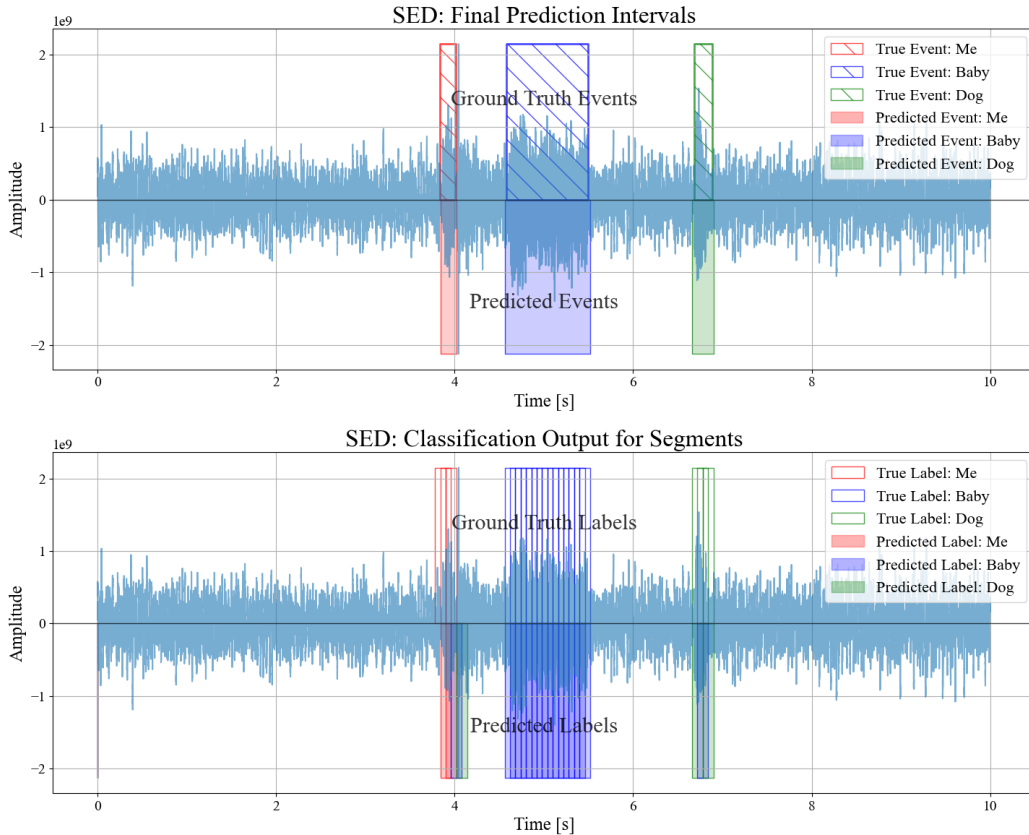


Figure 4.2: **Visualisation of the SED model output, Example I** - A 10 seconds audio file containing one event of each class; baby, dog and meerkat. (bottom) Plotting the true label segments over and the predicted segments under the x-axis. (top) The results of the post-processing. The true strong label over and the post-processed predicted segments, i.e the final event predictions, under the x-axis. "Me" represents the meerkat class.

4.3.2 Segmented Data

As the model is trained on small segments rather than full files, it can be a good idea to view the data as a set of segments rather than .wav files in order to understand the results better. In table 4.2, the number of segments for each class is shown for $\mathcal{D}_{0.2}^0$, separated by their ground truth label. Due to the segmentation of the events it is obvious that a big class imbalance arises. The difference is not due to any large discrepancy in event frequency, but rather a consequence of the difference in the distribution of event duration between the classes. The number of segments needed to cover an event is directly correlated with how long the event is. In figure 4.4, the number of event segments per file is plotted. This shows the distribution of how many event segments occur in files with events. In this plot, only the files with events are considered, as most files contain only noise. This information could be useful in order to distinguish files with events from files without events in an active learning context.

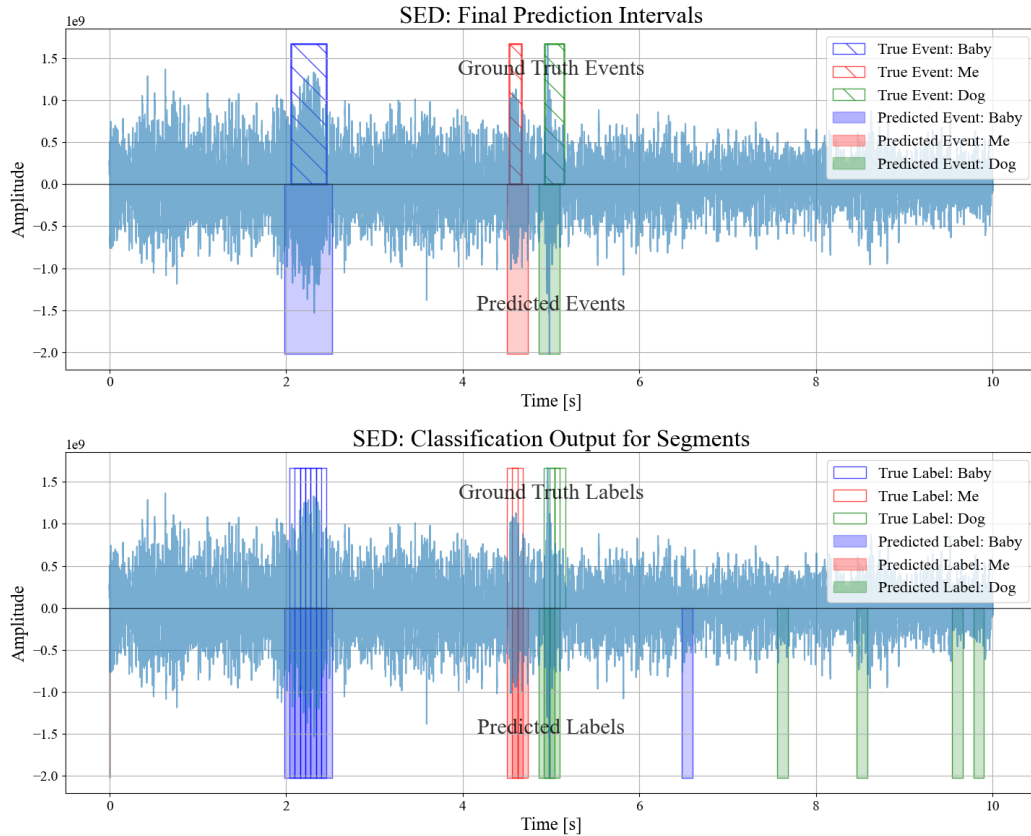


Figure 4.3: **Visualisation of the SED model output, Example II** - A 10 seconds audio file containing one event of each class; baby, dog and meerkat. (bottom) Plotting the true label segments over and the predicted segments under the x-axis. (top) The results of the post-processing. The true strong label over and the post-processed predicted segments, i.e the final event predictions, under the x-axis. "Me" represents the meerkat class.

Table 4.2: **Class Distribution in Segmented Data** - Distribution of classes after pre-processing. For the $\mathcal{D}_{0.2}^0$ data-set which is the main data-set used for studying active learning. "Me" represents the meerkat class.

Class	N. Segments	Percentage of data set	Average N. Segments per Event
Noise	429634	98.20 %	-
Baby	5048	1.15 %	16.7
Dog	1843	0.42 %	5.4
Me	975	0.22 %	2.87

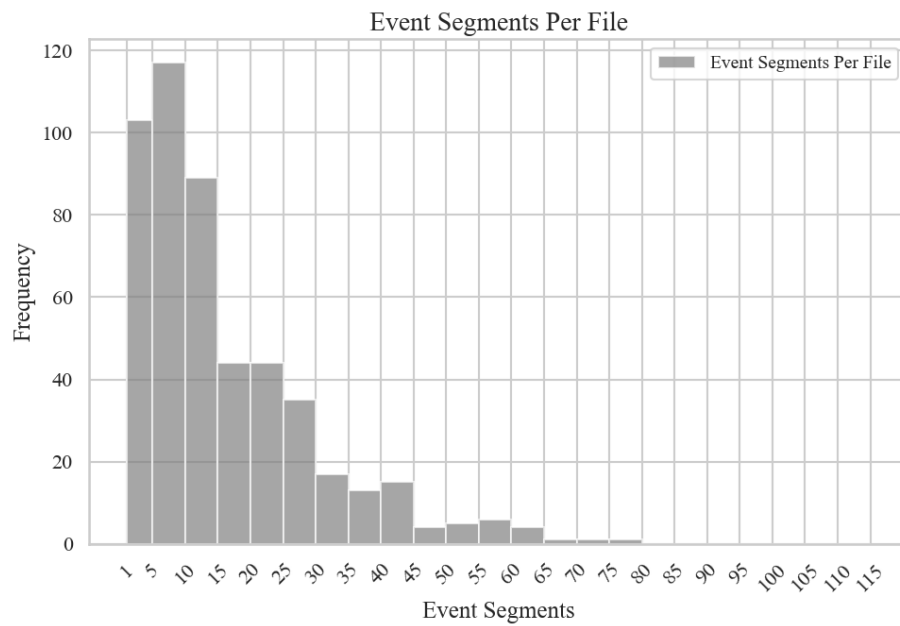


Figure 4.4: **Distribution of Event Segments in $\mathcal{D}_{0.2}^0$** - The distribution of the number of event segments present in the files in data set $\mathcal{D}_{0.2}^0$. Only files containing events are considered.

Chapter 5

Active Learning for SED

This chapter covers the methods used, the results produced and discussions of our study of active learning. Section 5.1 starts of by explaining how active learning is simulated and then continues with the setup used for the primary tests. This is followed by section 5.2 where the first results from testing different aggregation strategies is presented, along with discussions. Section 5.3 presents the result of the different diversification methods used and their relation to the aggregation strategies being used. Finally, the generalization of the strategies are presented in section 5.4. This is where new data sets are tested, in order to get an idea how robust the results are.

5.1 Method

5.1.1 Simulating Active Learning

In order to simulate active learning and test different strategies, one approach is to have access to a large annotated data set, where the size of the data set is more than enough to reach satisfactory results if all the data is used for training. The model is then initialized (trained) on a fraction of this data set, where the rest of the data gets defined as "unlabeled". The model can then iteratively access more and more data by asking for annotations from the "oracle" using a querying strategy . The oracle in this case is just simply revealing the labels previously hidden for the model. Alongside this, a baseline strategy is used, where queries are selected randomly, in batches of the same size as the active strategies.

Each strategy is assigned a model, which is retrained on a larger labeled set at each iteration. The model is evaluated on a separate validation/testing set, for valid performance results. By continuously increasing the amount of data accessible for the model, we can measure performance of the SED model in relation to the fraction of labeled data L_r , which is our budget. This is defined as:

$$L_r = \frac{|\mathcal{P}_L|}{|\mathcal{P}_L| + |\mathcal{P}_U|} \quad (5.1)$$

Here $L_r = 1$ means we use all the data allocated for training. These results can be used to visualise the performance improvement between the models trained on different amounts of data, given a querying strategy. This data can then be used to compare strategies, a good active learning method should see a quicker increase in performance with respect to L_r compared to the baseline strategy. The increase in data at each iteration can differ, but usually smaller batches, and thus less data, are queried in the beginning to emphasise the impact of active learning in the early stages of data gathering. This means that $|\mathcal{P}_Q|$ in algorithm 1 and algorithm 2 is not the same throughout. This is to allow for high resolution in the early stages, and to save computational resources in the later stages of active learning.

5.1.2 Setup

The initial focus of the active learning research is to verify that active learning is useful for SED and finding an aggregation method that gives a fast and significant increase in performance compared to the baseline strategy. Note that for all testing, a validation data set $\mathcal{D}_V \subset \mathcal{D}_r^{SNR}$ is used to evaluate

the model. The validation set consist of 20% of the files in \mathcal{D}_r^{SNR} . The same split is done for $\bar{\mathcal{D}}_r^{SNR}$ when testing active learning on a slightly different domain.

Table 5.1 presents how queries are chosen in terms of the number of files selected at each step. In the beginning, batches containing 10 queries are created, as we want to closely study what happens in the beginning of the learning process, as this is the crucial part of active learning, where more is to gain from a good strategy. We want to make sure that active learning is able to quickly gain an advantage over the baseline strategy, which is why we use a fine resolution in early stages of training. The baseline model queries files randomly instead of actively in all tests.

Table 5.1: **Active Learning Queries** - Summary of the number of files queried in the active learning loop at each iteration. L_r relates to the fraction of training files used and $|\mathcal{P}_L|_i$ indicates the size of the labeled pool at iteration i . The query sizes presented here will be used often throughout experiments, with a few exceptions. In some experiments, the active learning is stopped after iteration 12.

Iteration (i)	L_r	$ \mathcal{P}_L _i$
1	0.005	10
2	0.01	20
3	0.02	40
4	0.03	60
5	0.04	80
6	0.05	100
7	0.06	120
8	0.07	140
9	0.08	160
10	0.09	180
11	0.1	200
12	0.2	400
13	0.3	600
14	0.4	800
15	0.5	1000
16	1	2000

In the primary tests SNR is set to 0 and r is set to 0.2, which have been defined in section 3.1.1. The data set has size $|\mathcal{D}_{0.2}^0| = 2500$, whilst the validation data set has size $|\mathcal{D}_V| = 500$, as 20% is used for validation. The additional 2000 is kept for training. This is presented in table 5.2, along with the separation of the data set $\bar{\mathcal{D}}_{0.2}^0$, used for the final test. L_r is defined as the fraction of labeled data, with respect to the total files allocated for training, i.e 2000 files.

Table 5.2: **Data Setup** - The size and separation of data sets used for different tests of active learning.

Data Set	Size
$\mathcal{D}_{0.2}^0$	2500
\mathcal{D}_V	500
$\bar{\mathcal{D}}_{0.2}^0$	2500
$\bar{\mathcal{D}}_V$	500

When testing active learning, seeds are used to initialize the validation set \mathcal{D}_V and the first labeled data set, \mathcal{P}_L at iteration 1. This is to assure that all active learning strategies being compared have the same starting conditions. We run each experiment with a different seed to see if the same trends can be observed for different starting conditions. The seed are also used to initialize weights and mini-batches in the classifier-head so that if two strategies were to choose the same files to train on, the outcome would be the exact same. The results of all seeds can then be averaged to see the average outcome of active learning. This also allows for confidence intervals to be determined for all models. Five seeds are used; 2, 17, 41 91 and 118.

Other data sets

Besides looking at the generalization of the active learning models on a new domain, other contributing factors and parameters are studied. A data set $\mathcal{D}_{1.0}^0$ is created to make tests on how active learning behaves when all files contain events ($r = 1.0$). This is interesting for this particular field of study as event frequency might vary and it is important to establish what the impact of this change has on active learning results. Two additional data set $\mathcal{D}_{0.2}^{10}$ and $\mathcal{D}_{0.2}^{-10}$ are created where the SNR is different to see how this affects active learning. All tests executed on these data sets are performed in the same way as for the others, except they stop at iteration 12 in table 5.1.

5.2 Segment Aggregation Methods

5.2.1 Comparison of Aggregation Strategies

In figure 5.1, we present a comparison between the following aggregation strategies: *mean entropy*, *top 10 entropy*, *median entropy* and *mean event entropy*, which were introduced in section 2.2.3. These are compared to the baseline strategy, with respect to total IoU, a SED based metric, see section 2.1.4. This metric considers both the models ability to classify type of event and noise, and its precision in time. Note that these results are measured *after* filtering and post-processing the segment classifications into full events. The results are averaged over 5 seeds for reliability in the results. The baseline strategy is run 10 times per seed, due to its stochasticity. Note that the x-axis is not linear, but instead follows query sizes in table 5.1.

In the plot we can observe that both the *top X entropy*, with $X = 10$, and the *mean entropy* strategy perform a lot better than the baseline strategy. It appears that *top X entropy*, with $X = 10$, is the most beneficial active querying strategy. An additional observation can be made. When looking at the performance for the baseline strategy after querying 100% of the files, we see that the *top X entropy* reaches a similar performance when querying barely 8% of the data. This means that for that particular performance level, using active learning can cut the budget with 92%. The same goes for *mean entropy*, but where a similar performance level is reached around 9%. This is exactly the kinds of results we want to see when using active learning.

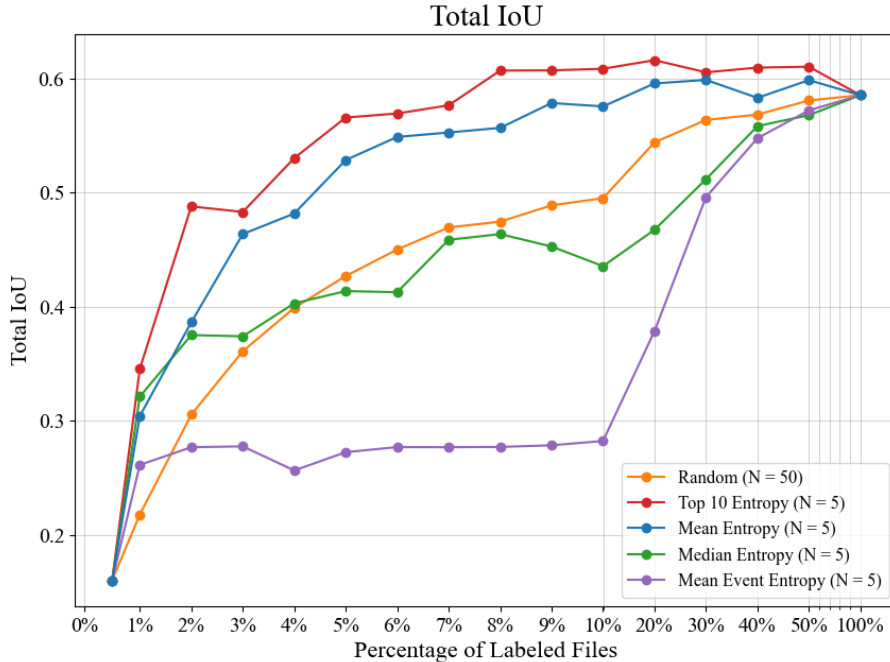


Figure 5.1: **Total IoU: Segment Aggregation Methods** - the total IoU results averaged over 5 seeds ($N=5$), for 4 different querying strategies (*mean entropy*, *top 10 entropy*, *median entropy* and *mean event entropy*) along with the baseline. This is computed after post-processing where the accumulated events are now compared with the true strong labels. Note that the x-axis is not linear, but instead follows query sizes in table 5.1. Vertical grid lines are spaced 10% apart. The baseline is run 10 times per seed.

To explain the results above we look at how the model performs on the raw predictions of event segments. In figure 5.2, the active learning results for the recall performance is shown, i.e classification accuracy for ground truth events, see equation (2.13). This means that performance for noise segments is ignored, as we don't consider TN or FP, see section 2.3.2. Note that we now consider the models raw classification output, *before* filtering and post-processing the segment classifications into full events. This metric is used as it isn't affected by the extreme class imbalance and shows how well the model classifies segments which has implications for the final SED output. The same aggregation strategies are studied in this case, and are averaged over the same 5 seeds.

The results here once again show that *mean entropy* and *top X entropy*, with $X = 10$, both outperform the baseline strategy. In this case we see that the *top X entropy* reaches the baseline performance at 100% after only querying barely 3% of the data. This reflects a budget reduction of 97%. The same goes for *mean entropy*, but where a similar performance level is reached around 4%. Additional observation is that the *median entropy* also outperforms the baseline in this case. The *top 10 entropy* strategy is not the single most beneficial strategy when considering this metric. In general the model seem to improve even more using active learning in this case. This can be explained by either the fact that the post-processing has flaws or that some strategies give poor performance for classifying noise, as this is not considered in this metric. Let's look at the overall accuracy to examine this further.

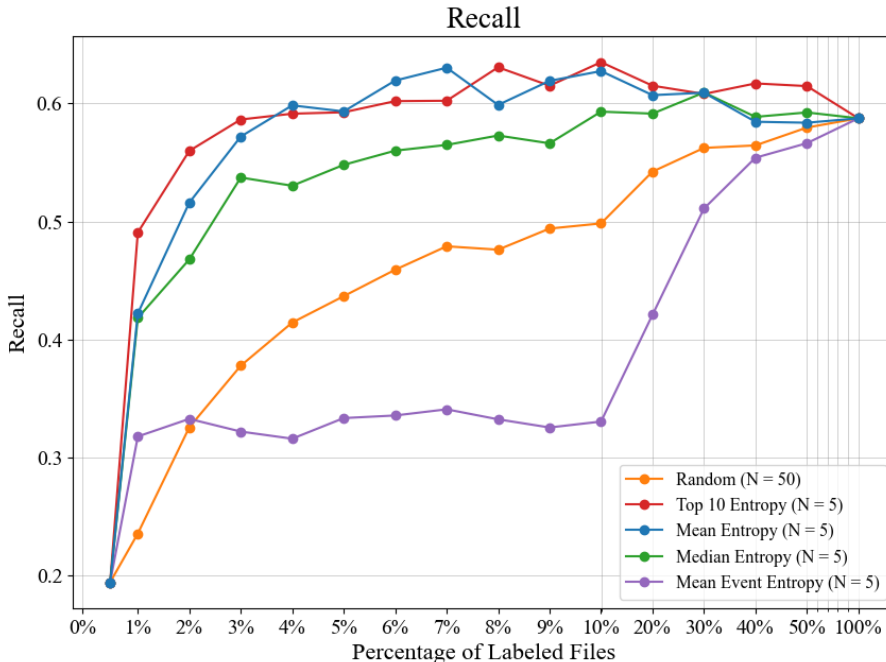


Figure 5.2: **Recall: Segment Aggregations Methods** - The averaged recall results over 5 seeds ($N=5$), for 4 different querying strategies (*mean entropy*, *top 10 entropy*, *median entropy* and *mean event entropy*) along with the baseline strategy. This metric only considers event segments, i.e how well the model classifies segments that contain animal vocalizations. Note that the x-axis is not linear, but instead follows query sizes in table 5.1. Vertical grid lines are spaced 10% apart. The baseline is run 10 times per seed.

In figure 5.3, the active learning results for the normal definition of accuracy is shown, see equation (2.11). Note that this also is measured *before* post-processing is performed. The difference from figure 5.2 is that the noise segments are now included in this metric. Accuracy is a bit biased in this case, considering the extreme class imbalance seen in table 4.2. This metric is dominated by noise segments, meaning that the model’s ability to classify events is not captured by the numerical values produced by the metric. This makes this metric not very informative, as it focuses on noise. An interesting result is that *top X entropy*, with $X = 10$, appears to have the best performance when considering noise, compared to the other active learning strategies. It does not however outperform the baseline initially.

This can explain the difference in total IoU and recall performance previously discovered, between the top performing strategies. Figure 5.3 shows that *mean entropy* lacks in performance in general accuracy, i.e. performs poorly on classifying noise. This fact is not captured by the recall metric but is considered in total IoU. This also explains the reason why *median entropy* performed relatively well on recall compared to total IoU. It is clear that both events and noise are important for good SED performance, and *top X entropy* is the strategy that manages both well.

We can establish that the aggregation strategies most successful so far are *top X entropy*, $X = 10$, followed by *mean entropy*. To assure the significance in the performance, confidence intervals are created using the realisations from 20 seeds. In figure 5.4 the total IoU, along with matched F_1 -score, averaged over all seeds are plotted for *mean entropy* and *top 10 entropy* with 95 % confidence intervals. Matched F_1 -score is an additional SED-metric, more focused on the temporal aspect compared to total IoU, see section 2.1.4. The intervals for *top X entropy* and the baseline are separated for all budgets. Thus, we can say with 95 % confidence that *top X entropy*, with $X = 10$, is better than the baseline strategy for the metrics studied. This is not the case for the *mean entropy* strategy when the matched F_1 -score is considered, but hold for almost all budgets in the total IoU case. An additional mark is that the *top X entropy*, with $X = 10$, performs significantly better than the *mean entropy*.

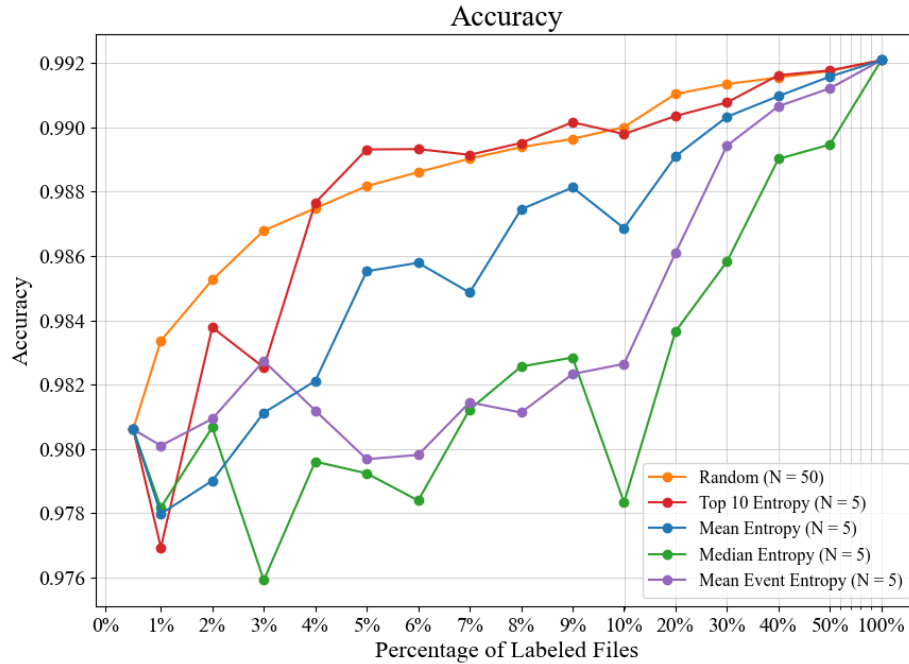


Figure 5.3: **Accuracy: Segment Aggregations Methods** - The accuracy results averaged over 5 seeds ($N=5$), for 4 different querying strategies (*mean entropy*, *top 10 entropy*, *median entropy* and *mean event entropy*) along with the baseline strategy. This metric considers all segments, including noise. Note that the x-axis is not linear, but instead follows query sizes in table 5.1. Vertical grid lines are spaced 10% apart. The baseline is run 10 times per seed.

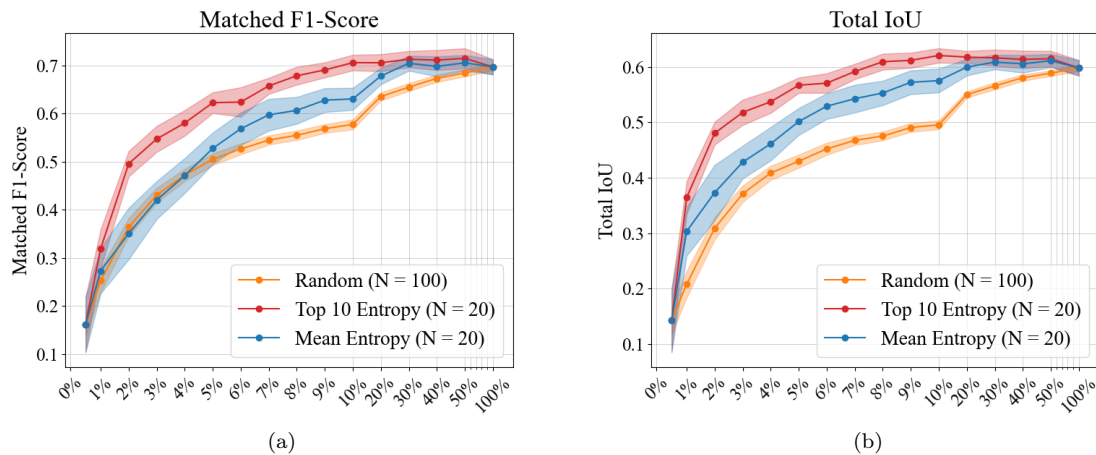


Figure 5.4: **Matched F_1 -score and Total IoU: Segment Aggregation Methods with Conf. Intervals** - The (a) matched F_1 -score and (b) total IoU results averaged over 20 seeds ($N=20$), for 2 different querying strategies *mean entropy* and *top 10 entropy* along with the baseline. The orange line represents the average baseline strategy over 5 realisations for each seed and is displayed with 95% confidence intervals. The same goes for the querying strategies, where the 95% confidence intervals is based on the average of seeds.

5.2.2 Analysis of Queried Files

It can be useful to inspect what type of queries the different active learning strategies make, to get a better understanding of the outcome of the results. Considering that there is an extreme class imbalance in favor of noise segments, and that the event segments are important to detect for the purpose of SED, it is likely that the SED would improve if queries are made for files that contain events. As these files also are made up of a majority of noise (98.2%), the model will still likely perform well at classifying noise. There should therefore not be any big downside with querying files with events, with an upside being that more data to support the event predictions are provided. It is thus reasonable to assume that a good querying technique should choose files with events being present. In this section we check if this hypothesis holds true, and what implication the queries have for the performance.

In figure 5.5 the average number of segments queried at each iteration is shown for the baseline model and the models using the strategies *top 10 entropy*, *mean entropy*, *mean event entropy* and *median entropy*. The classes are color coded, in order to distinguish how many queried segments belong to each class. These results stem from the same five seeds as before. Iteration 1 corresponds to the number of event segments the initial data set consists of and is the same over all models. The later iteration tends to consist of more event segments as these contain a larger percentage of the data set, thus more files, compared to the earlier iterations.

It is apparent that the aggregation strategies that perform better, being *top 10 entropy* and *mean entropy*, query files with more event segments, opposed to *mean event entropy* or *random*. This can also explain the initial loss in overall accuracy for some of the active learning strategies in figure 5.3. When querying files containing events, the dominant class, noise, gets neglected. As noise makes up the vast majority of all predictions, it has largest repercussion on the accuracy metric.

As previously mentioned, it is interesting to look at the initial stages of querying as this is where active learning is crucial. Table 5.3 shows the percentage of event segments queried when 2% of the data has been annotated along with the models recall and total IoU performance at that iteration. As 2% of the files have been queried, we can expect the baseline strategy to sample 2% of events, as it doesn't actively seek out events. This is also the observed outcome. The difference is drastic compared to *top 10 entropy* and *mean entropy*, where 14% and 10% of all segments are found in 2% of the data. The same results for 10% of the data is presented in table 5.4, where it is shown that 61% and 43% of events are found in 10% of the data for *top X entropy* and *mean entropy* respectively. This is clear evidence that these strategies actively query files with events, and that this pays off in terms of recall and total IoU. There appears to be diminishing returns in this payoff, especially for recall. This can be seen as the difference between the *mean entropy* and *top X mean entropy* is large in terms of queried segments, but small in performance, whereas when comparing the baseline to *mean entropy* the differences are large for both metrics. This means that we expect less increase in performance per queried event as we query more and more events. In general, there appears to be grounds for the claim regarding queried events being related to performance.

Table 5.3: **Queried Event Segments (%) - Iteration 3** - The total fraction of queried event segments to the total number of event segments in the data set. This is a snapshot of the active learning loop at iteration 3, where 2% of the training data has been queried. This data is accompanied by two of the performance metrics: total IoU and recall. These are the average results over 5 seeds. Random is run 10 times per seed.

Event Type	Random	Top 10	Mean	Mean Event	Median
Baby Segments	2%	16%	12%	1%	5%
Dog Segments	2%	12%	7%	3%	3%
Meerkat Segments	2%	7%	5%	2%	4%
All Event Segments	2%	14%	10%	2%	5%
Recall	0.33	0.56	0.52	0.33	0.46
Total IoU	0.31	0.49	0.38	0.28	0.37

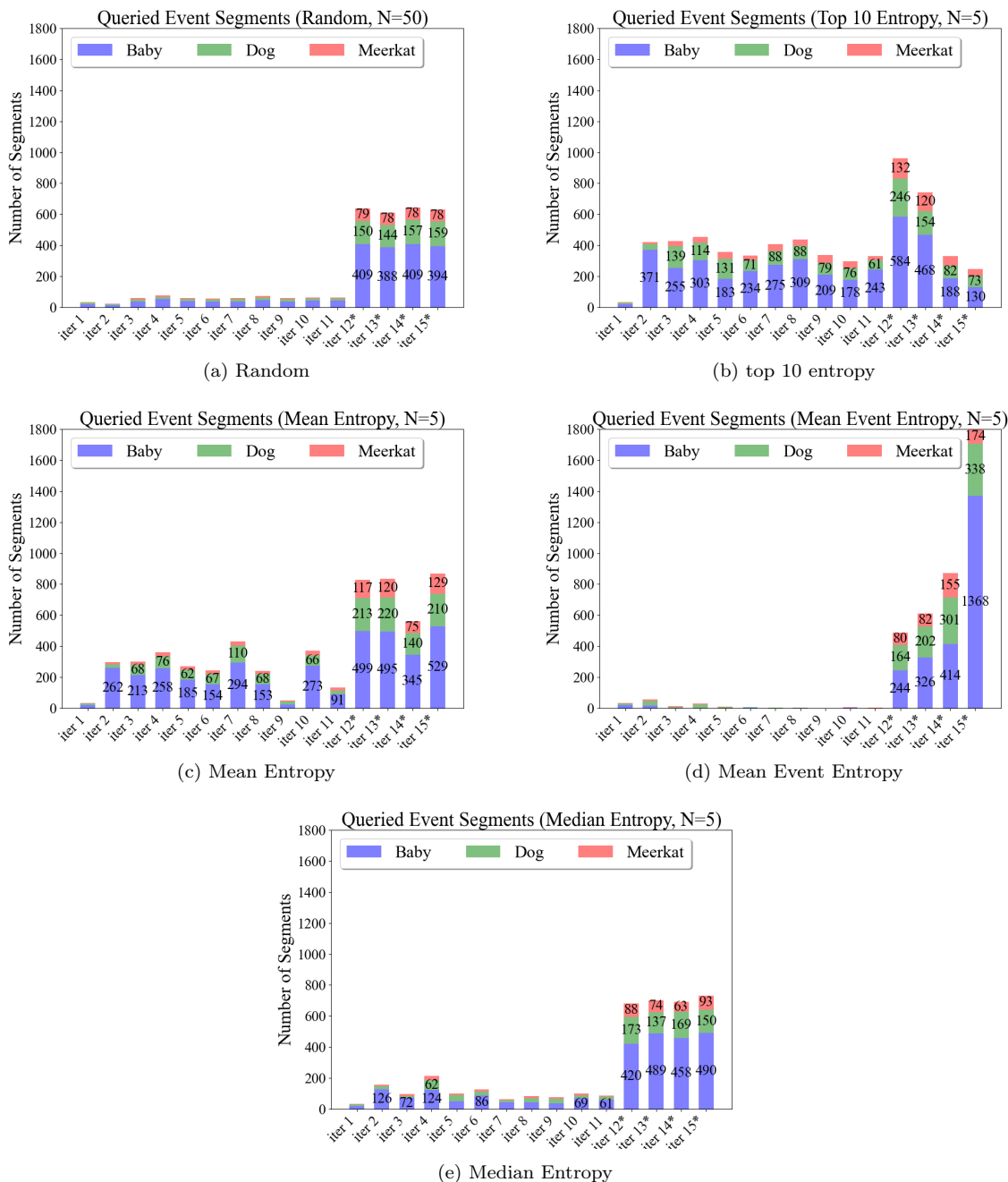


Figure 5.5: **Queried class segments: Segment Aggregation Methods** - The number of queried event segments separated by class at each iterations for (a) *random* (baseline strategy) and the querying strategies: (b) *top 10 entropy*, (c) *mean entropy*, (d) *mean event entropy* and (e) *median entropy*. These are averaged over 5 seeds. Random is run 10 times per seed.

Table 5.4: **Queried Event Segments (%) - Iteration 11** - The total fraction of queried event segments to the total number of event segments in the data set. This is a snapshot of the active learning loop at iteration 11, where 10% of the training data has been queried. This data is accompanied by two of the performance metrics: total IoU and recall. These are the average results over 5 seeds. Random is run 10 times per seed.

Event Type	Random	Top 10	Mean	Mean Event	Median
Baby Segments	10%	64%	48%	1%	18%
Dog Segments	10%	60%	40%	6%	19%
Meerkat Segments	10%	47%	28%	4%	17%
All Event Segments	10%	61%	43%	3%	18%
Recall	0.50	0.63	0.63	0.33	0.59
Total IoU	0.50	0.61	0.58	0.28	0.44

When looking at the class specific data in tables 5.3-5.5, keep in mind that the results show the ratio of queried segments within that class. It is clear that there are some differences between classes. For the more successful querying strategies, *top 10 entropy* and *mean entropy*, babies are over-represented, whereas meerkats appear to be more difficult to query. Keep in mind that there is a class imbalance in play, which means that these differences will be even bigger in absolute terms.

In table 5.5, the results preceding the final iteration (15) of the active querying is shown, corresponding to 50% of the data. We see how many of the event segments in the data set that are queried by each strategy. As this is the last iteration where active queries are made, this result tells us how many event segments the different strategies find in total, and how many event segments that the active strategies leave behind. Whereas *top 10 entropy* and *mean entropy* manage to query 95% of event segments, the other active strategies query way fewer events. At this stage in the training, this doesn't seem to have a big impact on the performance metrics, as they are comparable. It is plausible that the model is saturated at this point. It is clear from table 5.3 that in earlier iterations, querying event segments leads to a leap in performance. At some stage, the model benefits less from being presented to events. The model might have seen enough variability for the events, and thus struggles to improve further. Figure 5.2 shows us a slight decrease in recall in the later iterations for the better performing aggregation strategies. This could potentially be due to the model no longer benefiting from the exposure of event segments, but still needs to accommodate more and more noise during training. We clearly see that more is to gain from active learning in the early stages, as the model appears more reactive to the data.

Table 5.5: **Queried Event Segments (%) - Iteration 15** - The total fraction of queried event segments to the total number of event segments in the data set. This is a snapshot of the active learning loop at iteration 15, where 50% of the training data has been queried. This data is accompanied by two of the performance metrics: total IoU and recall. These are the average results over 5 seeds. Random is run 10 times per seed.

Event Type	Random	Top 10	Mean	Mean Event	Median
Baby Segments	50%	98%	95%	60%	64%
Dog Segments	51%	97%	92%	73%	61%
Meerkat Segments	50%	93%	85%	67%	58%
All Event Segments	50%	97%	93%	64%	63%
Recall	0.58	0.61	0.58	0.57	0.59
Total IoU	0.58	0.61	0.6	0.57	0.57

5.2.3 Finding a Suitable X in $top X$ entropy

In the initial testing, the $top X$ entropy aggregation method showed promising results as seen in figures 5.1-5.5. However, the X in $top X$ entropy was set to 10 quite arbitrarily. As each file is made up of 175 segments, $mean$ entropy corresponds to $top X$ entropy with $X = 175$. As we can see a difference in results between the two, there is reason to believe that the selection of X matters. In this section, the selection of X is discussed in detail, and some new strategies that extend the $top X$ entropy strategy are tested.

Fixed X

We will start of with keeping X fixed through out the entire training process and test for different numerical values. To choose some valid candidates, the definition and restrictions of X has to be established. The following is true for X :

$$X \in \{x \in \mathbb{Z}^+ \mid 1 \leq x \leq 175\}$$

The restriction is due to the number of segments present in each file, where X cannot exceed this value. Figure 5.6 shows the result of testing different fixed values of X in an active learning context. The testing and results is based on the same methodology as before, where the same 5 seeds are used. The values being studied are $X \in \{2, 5, 10, 20, 40, 80\}$.

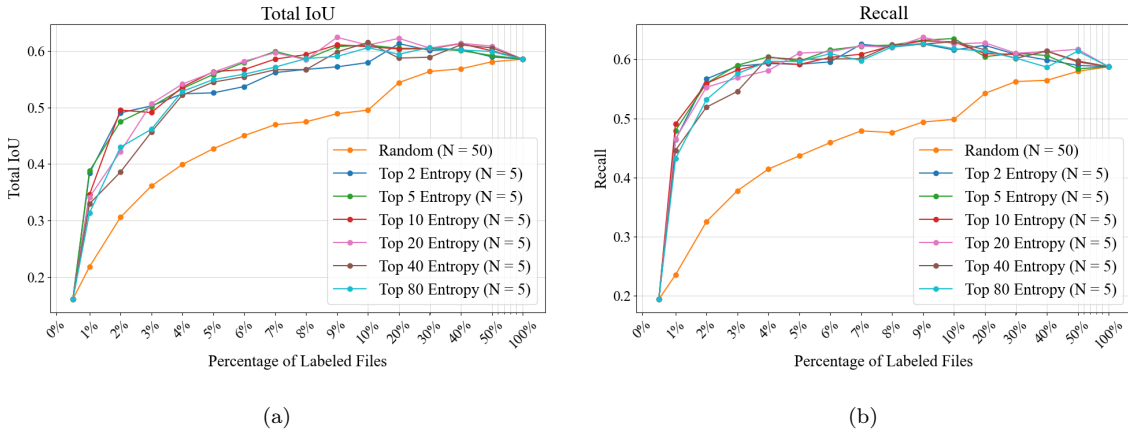


Figure 5.6: **Total IoU and Recall: Different $top X$ entropy** - The (a) total IoU and (b) recall results averaged over 5 seeds ($N=5$), for 6 different fixed values of X for the querying strategy ($top X$ entropy) along with the baseline strategy. The values being studied are $X \in \{2, 5, 10, 20, 40, 80\}$. The baseline is run 10 times per seed.

It is apparent that all different fixed values of X outperform the baseline, but any differences between them is hard to distinguish. In the early stages it seems that a relative small value of X seem to perform slightly better for both of the metrics. This is consistent with the results presented in the previous section where we have seen that $X=10$ outperform $X=175$ (mean entropy) for small budgets. For total IoU it seems that the lowest value, $X = 2$, manages to have a quick increase at the start, but eventually starts to fall behind after about 5%. This could mean that a X that varies with the percentage of labeled files L_r is beneficial, as the model changes which affects its output. Further, it could be of interest to study how the accuracy for the individual classes, to see how this could be affected by the choice of X . The reasoning behind this is that the events look different in terms of average number of segments per event, see table 4.2.

Figure 5.7 shows the total IoU for each class and all the different top X strategies, along with the baseline strategy. It seems that a low value of X favours the meerkat class in the beginning of the process, whereas these values perform poorly for the baby class. This can be explained by the difference in the number of segments an event of the difference classes contain. The value of X that correlates to

a standard length of an event of a certain class might favour the querying of files containing that class. Whilst we can find arguments that supports such a conclusion, these results seen in 5.6 and 5.7 are by no means significant. When studying these X values, it is seldom that the observations mentioned here hold for the entire plot. Hence, these observations should be taken with a grain of salt.

The idea that the *top X entropy* with a low X value might favor classes with shorter events, could be explained by how the segment based SED method works. If a class has short events, there will be a class imbalance: the model see fewer segments from that class. This could in turn lead to more uncertainty for this class, and as the events are of length X , the *top X entropy* would be high and not get washed out by more certain predictions of other classes. With that said, this assumption might not hold if the short class is easier to predict, or if there are multiple events in the audio files as there is in this case.

The observations made from studying *top X entropy* and the distribution of event lengths seen in figure 4.4, lead us to believe that there is more to explore with this strategy. Enter, the *top X ensemble*.

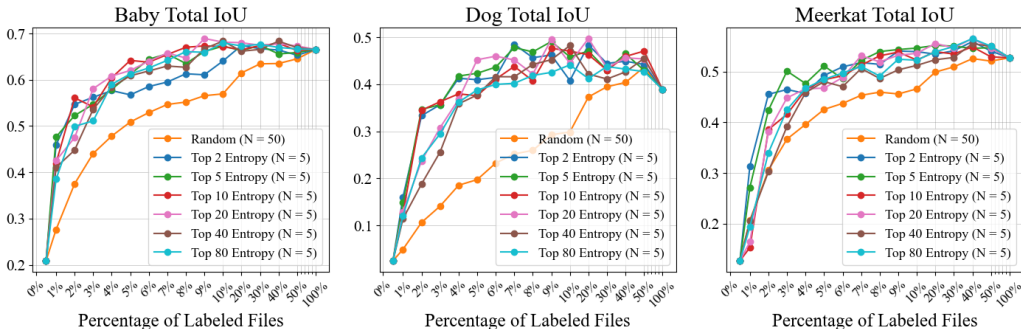


Figure 5.7: **Total IoU on Classes: Different *top X entropy*** - The Total IoU performance on each individual class averaged over 5 seeds ($N=5$), for 6 different fixed values of X for the querying strategy (*top X entropy*) along with the baseline strategy. The values being studied are $X \in \{2, 5, 10, 20, 40, 80\}$. The baseline is run 10 times per seed.

An ensemble of top X entropy

We have seen some evidence that different *top X entropy* measures are better or worse at different classes. It is reasonable to suspect that this has something to do with the correlation between X and how many event segments are found in a file. The reasoning behind this is based on that a lower X performs better for meerkat events, and worse for baby events. We have also previously noted that meerkat segments are much rarer than babies. Therefore, when choosing a batch, it could be reasonable to use multiple values of X . Each X can then be used to select a number of files for the query, using *top X entropy*. If successful, this strategy might find many events based on a slightly different approach, hopefully introducing a bit of diversity so that all three classes show good performance.

We will begin by defining an ensemble of X values, or a *top X ensemble*, before we explain how the ensemble is selected. An ensemble $S_X = X_1, \dots, X_K$ is a set of values which are used in the *top X entropy* querying strategy. Each value $X \in S_X$ is used to query a fraction of files in each iteration, where each X chooses the same number of files if possible.

Let's give an example. We are on iteration 12 and are about to query a batch of 10% of the unlabeled data set, this corresponds to 200 files. A set of 10 integers are used in an ensemble, for instance $S_X = \{10, 20, 30, 40, \dots, 100\}$. When using this ensemble we start of by letting *top 10 entropy* choose 20 files, i.e a tenth of the total queries being made. Then we let *top 20* query an additional 20 files (the previous 20 files are not considered, as they are already in the batch). This continues until all 10 values in the set has chosen 20 unique files, making up a full batch of 200 files. This is repeated at each iteration, potentially with a new ensemble.

Whilst the results were not significant in figure 5.7, we still suspect that some *top X* values benefits the training of specific classes. We want to create an ensemble that accommodates all classes at once. We design this ensemble around the assumption that a value of X is good at querying files containing X event segments. This means that sampling from the distribution which determines how many segments are found in a file, gives a good distribution of ensemble members. As previously seen in figure 4.4 the underlying distribution of event lengths (in terms of segments) can be established. Thus, by fitting a probability density function to this data we can sample the number of segments, and use these samples as X values. This means that the method relies on prior knowledge of the distribution. In this experiment, perfect knowledge is assumed, meaning that all true labels in $\mathcal{D}_{0.2}^0$ have been used for the distribution, rather than estimating this on the labeled pool \mathcal{P}_L on each iteration. The probability density function used is a shifted gamma distribution:

$$f(x; \alpha, \beta, \text{loc}) = \frac{1}{\beta^\alpha \Gamma(\alpha)} (x - \text{loc})^{\alpha-1} e^{-(x-\text{loc})/\beta} \quad (5.2)$$

$$\alpha = 1.1712,$$

$$\beta = 12.5849,$$

$$\text{loc} = 0.9927.$$

The result of the fitting is shown if figure 5.8. We can now sample from this distribution, and use the samples in the *top X* ensemble. This means that \mathcal{S}_X contains 10 different X sampled from the distribution in (5.2). **At each iteration** a new ensemble \mathcal{S}_X is sampled, so that the ensemble is not to be biased towards one single realisation. Given that we have introduced stochasticity, it is a good idea to run the strategy multiple times for each seed.

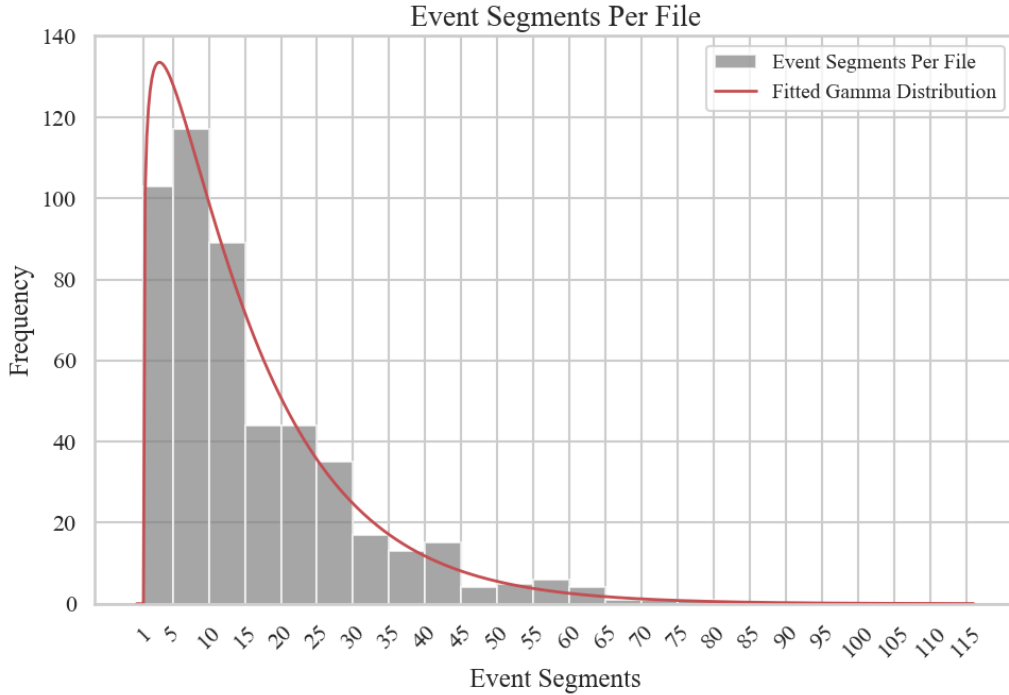


Figure 5.8: **Fitted Gamma Distribution of Event Segments in $\mathcal{D}_{0.2}^0$** - A fitted gamma distribution curve for the distribution of the number of event segments present in the files for data set $\mathcal{D}_{0.2}^0$. Only files containing events are considered. The exact distribution and the value of parameters used is shown in equation (5.2).

Figure 5.9 shows the result of the ensemble as querying strategy along with the previously used *top 10 entropy* and the baseline strategy as reference. The tests are executed in the same procedure as before, with the same 5 seeds. The baseline strategy is run 10 times for each seed, whereas the ensemble is run 5 times per seed. Unfortunately, the figure reveals that no significant improvements were made using this strategy. Though, the method is worth mentioning as it can be more thoroughly examined. The number of values sampled at each step was arbitrarily chosen to be 10, and is an additional hyperparameter that can be tuned. An additional thing is that the distribution of events in the files can affect this method. The ensemble aims to uniformly query each class based on the number of event segments in a file. This is not accomplished, as seen in table 5.6, where the distribution of queried class segments queried does not change much compared to *top 10 entropy*. This can be explained by the fact that many of the event files contain multiple events. The sum of event segments in those files are hard to assign to specific class, as they can contain multiple different events. Thus, this ensemble strategy could benefit if run on a data set containing event files with only one single event.

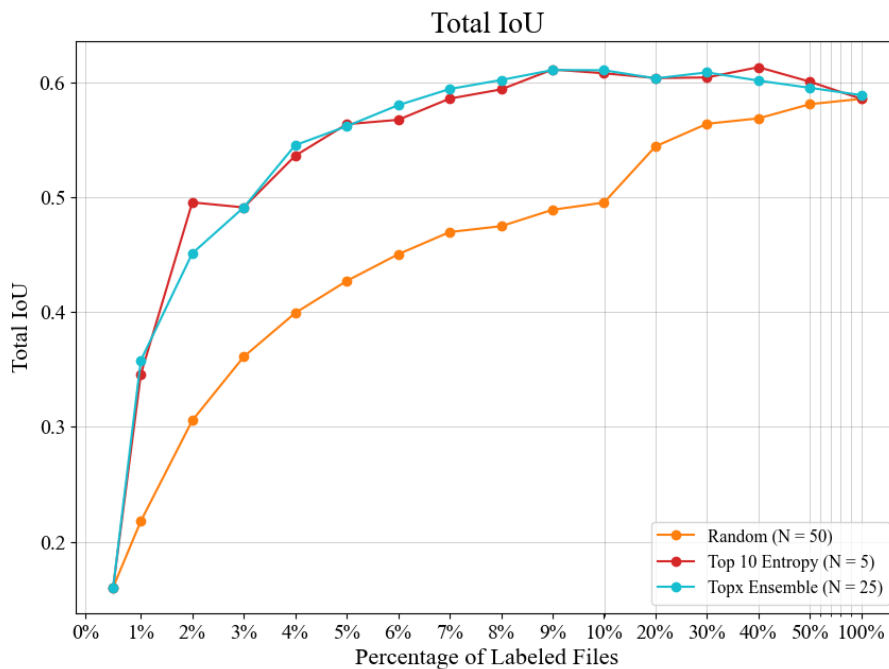


Figure 5.9: **Total IoU: Ensemble for *top X entropy*** - The averaged total IoU results over 5 seeds ($N=5$), for the querying strategy using a *top X* ensemble compared with *top 10 entropy* and the baseline strategy. The baseline is run 10 times per seed, whereas the ensemble is run 5 times per seed.

Table 5.6: **Queried Event Segments for Ensemble (%) - Iteration 11** - The total fraction of queried event segments to the total number of event segments in the data set. This is a snap-shot of the active learning loop at iteration 11, where 10% of the training data has been queried. This data is accompanied by two of the performance metrics: total IoU and recall. These are the average results over 5 seeds. The two aggregation strategies being compared are *top 10 entropy* and *top X ensemble*

Event Type	Top 10	Top X Ensemble
Baby Segments	64%	62%
Dog Segments	60%	59%
Meerkat Segments	47%	49%
All Event Segments	61%	60%
Recall	0.63	0.64
Total IoU	0.61	0.61

5.3 Diversification

Given the aggregation strategies from the previous section, we now attempt to diversify the queries in an attempt to further improve on the active learning framework. This means that the files in the batch being queried should be different with respect to some similarity measure. We no longer make queries solely based on aggregation strategies, instead the task is divided in to two parts. Firstly, the aggregation strategy queries files that the model is uncertain of, the diversification strategy then fine tunes this selection by selecting files that differ from each other. This is performed by selecting a larger pre-batch based on aggregated uncertainty, and then select the final queries using a diversification strategy. The diversification strategies studied are random selection and the farthest traversal principal.

5.3.1 Random Selection

Aggregation strategies is biased in its ways of choosing files, as it is deterministic. Therefore, we let the aggregation strategies select a larger number of files and then we randomly select from these. The large batch still contains valid contestants as it is based on uncertainty, but the adding of stochasticity in the form of random sampling, bias is reduced. Thus, a more diverse reflection of the data is retracted.

5.3.2 Farthest Traversal

Farthest traversal is an active way of deciding which files are actually queried, by selecting files that are dissimilar. In order to use farthest traversal as in algorithm 3, an embedding that represents entire files are necessary. Previous embeddings have been created on short segments using YAMNet. Using YAMNet in this case would require comparing many small segments with each other, which would require a lot of computation, and might not be valid representations of the entire files. For a more straight forward comparison between files, we opt for a different embedding model that more effectively embeds larger segments of audio, with a larger dimensionality reduction. Two models that allow for 5 and 10 second audio respectively are *Perch* and *AudioMAE*, see section 2.1.3. When using these to embed files, necessary resampling is done to make the full 10s files compatible with the model. We test both models in order to see if there is any difference in how they represent the 10s files, which could have implications on the results.

Visualising full audio files

A *t-distributed stochastic neighbor embedding (t-SNE)*, of the embeddings produced by both models is performed for visualisation of the two models ability to grasps the underlying patterns in the data. t-SNE is an algorithm that reduces the dimensions while retaining salient patterns of complex and high-dimensional data. This is done in order to visualise vital clusters and underlying structures.

Figure 5.10 shows a 2-dimensional representation of the 1280-dimensional embeddings obtained from Perch. Each file embedding is color coded for which type of events that file contains. The black dots represents the files that are solely noise. An interesting thing that can be seen in the plot is that it seems that Perch manages to cluster noise files in a good way. The fact that the noise files themselves are separated into multiple clusters can be explained by the generation of data. It might be that each cluster consists of files sharing the same noise, as these are randomly generated from a discrete set of background audio files, and thus should contain the same features.

Looking at the t-SNE visualisation for the AudioMAE embedding similar conclusions can be made, see figure 5.11. Figure 5.12 and 5.13 shows how different classes are interpreted in the embeddings produced by both models. Both plots only confirms that both models seem to focus on finding patterns in the noises rather than in the events. This is not necessarily a bad thing. Diversification of noise can be useful as the noise itself has variability, and it can help teach the model to classify similar events in different contexts. However, we are more interested in improving class performance, meaning we are probably more interested in a good representation of files with events. The clusters of what is expected to belong to the same noise tend to be surrounded by a few dots that contain events. This is mainly a bad result, as it probably means that a files position in the high dimensional space is mostly determined by its noise, rather than the events in contains. However, the embeddings still show some sense of awareness to events. Ideally, this plot would show 8 distinct clusters, separated by which classes occur in the file. Even though this is very far from the case there is still some potential in the embedding representation.

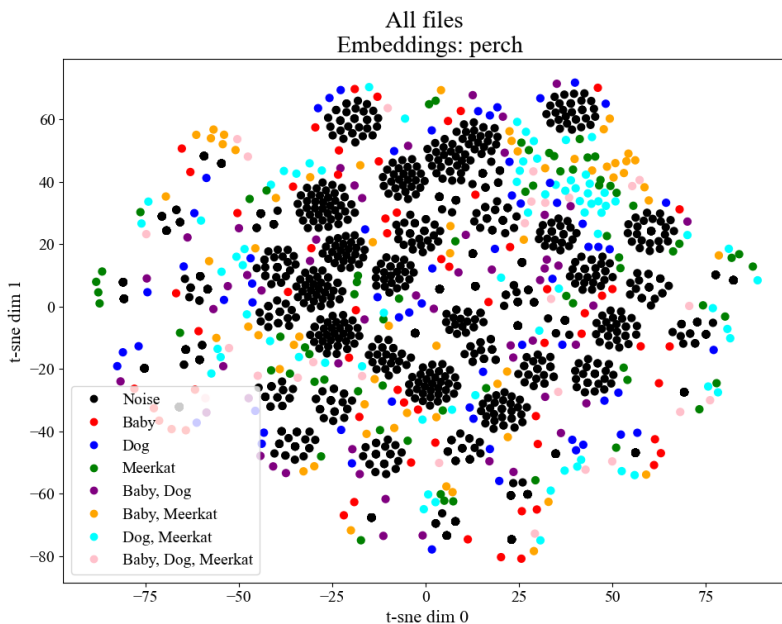


Figure 5.10: **t-SNE Visualisation - Perch** - A t-SNE representation of the 1280-dimensional embeddings produced by Perch. The color coding is based on the event types being present in the file, where black means that no events occur in the file.

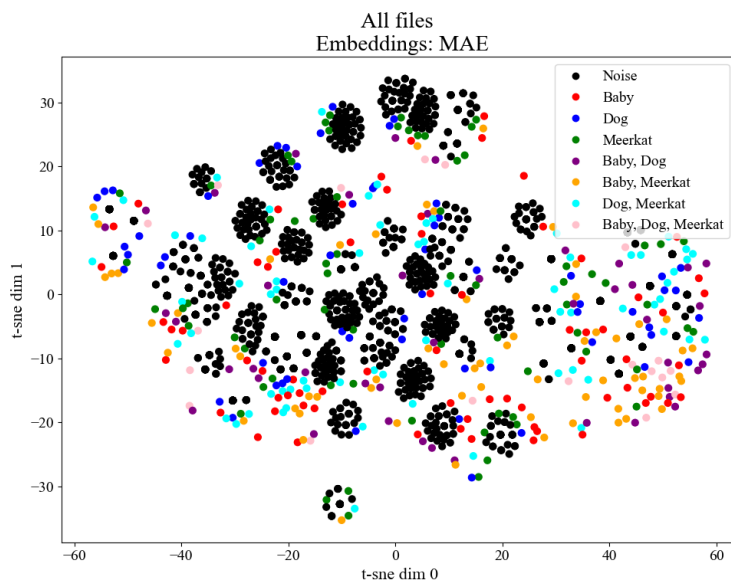


Figure 5.11: **t-SNE Visualisation - AudioMAE** - A t-SNE representation of the 768-dimensional embeddings produced by AudioMAE. The color coding is based on the event types being present in the file, where black means that no events occur in the file.

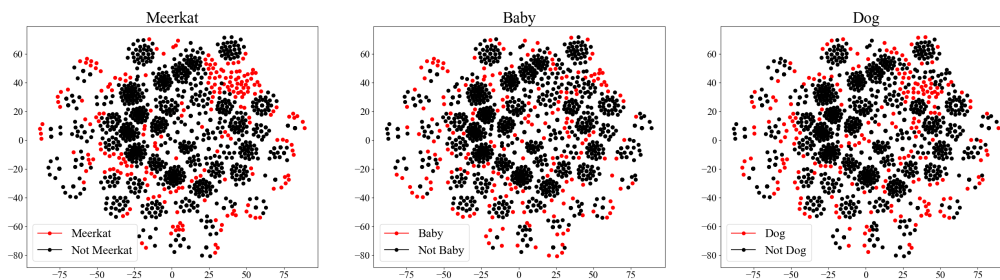


Figure 5.12: **t-SNE Visualisation - Perch (classes)** - t-SNE representations of the 1280-dimensional embeddings produced by AudioMAE for each class. The color coding is binary, where red means that the class in question occurs at least once in that file

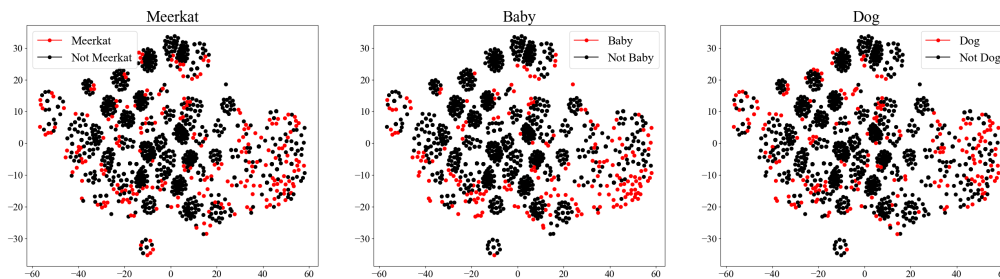


Figure 5.13: **t-SNE Visualisation - AudioMAE (classes)** - t-SNE representations of the 768-dimensional embeddings produced by AudioMAE for each class. The color coding is binary, where red means that the class in question occurs at least once in that file.

Setup and Results

Before performing farthest traversal using the two different embedding methods, and random selection, the size of the pre-batch has to be determined. This size determines the number of files that the diversification methods are allowed to choose from at each iteration. We chose to double the batch size, which we denote with the scaling constant $S = 2$. Thus the diversification strategy queries half of the files in the pre-batch.

Figure 5.14 shows the recall and total IoU result of the diversification strategies when combined with the *top 10 entropy* aggregation strategy. The performance of *top 10 entropy* from previous tests is also included for comparison. In the recall case we can see that farthest traversal using AudioMAE embeddings seem to outperform all models in the early iterations, but the difference is slim and not statistically significant. What is interesting though is looking at the diversification strategies impact when combined with other aggregation strategies. Looking at same performance metrics for *median entropy* with diversification in figure 5.15, the difference is more significant. In the total IoU case both farthest traversal strategies performs better than the other strategies. In the recall case, and in total IoU, we can see that random selection performs relatively poorly. The reason for why *median entropy* sees a decrease in performance with random diversification could be due to the fact that median entropy itself is a quite bad aggregation method. Thus, random sampling from a larger batch of poor chosen queries might just deteriorate the querying. It is also interesting that Farthest traversal performs so well in this case compared to the previous, where *top 10 entropy* was used. The reason why *median entropy* sees an increase in performance with the diversification strategies could originate from the same reason: that *median entropy* is a bad aggregation strategy to begin with. There are active learning frameworks that aren't uncertainty based, but only use methods that are based on similarity. Diversification in itself can be enough to make active queries that benefit results, and that might be what we are seeing here.

The same reasoning could apply to why the *top 10 entropy* fails to improve, it might already choose diverse queries, or optimal queries, which is why diversification doesn't make a big difference in either direction. Given each seed, there exist an optimal selection of files. If the selection of queries is perfect, then the results are saturated and diversification can do no good. Whilst there is no clear evidence to support that this is the case here, it could be worth keeping in mind. Furthermore, as the batch sizes change throughout the active learning loop, it is also possible that the hyper parameters of the diversification strategy should change accordingly.

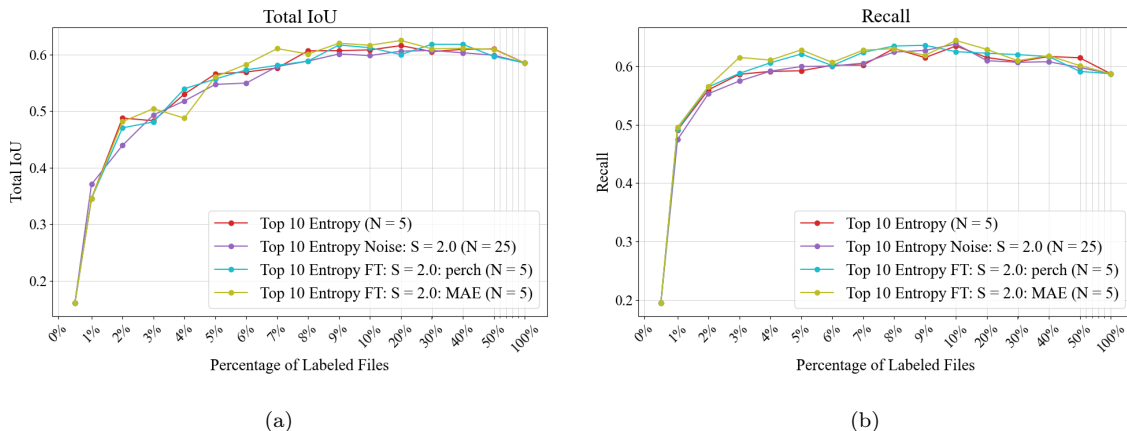


Figure 5.14: **Total IoU and Recall: Farthest Traversal *Top X Entropy*** - The (a) total IoU and (b) recall results averaged over 5 seeds ($N=5$), for the querying strategy *top X entropy* using the diversification strategies: random selection, farthest traversal and along with not diversification. The farthest traversal strategy is split in two, where two different embedding methods are used; Perch and AudioMAE. The random selection strategy is run 5 times per seed.

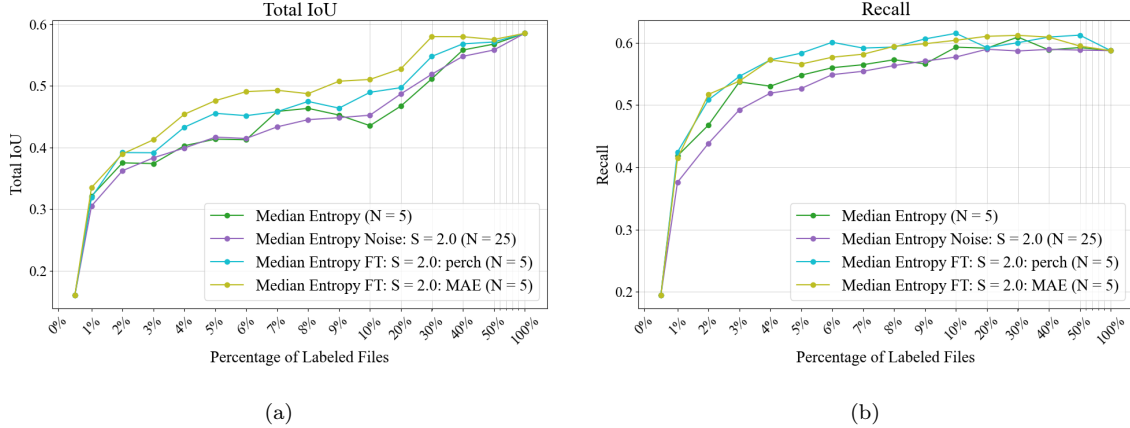


Figure 5.15: **Total IoU and Recall: Farthest Traversal Median Entropy** - The (a) total IoU and (b) recall results averaged over 5 seeds ($N=5$), for the querying strategy *median entropy* using the diversification strategies: random selection, farthest traversal and along with not diversification. The farthest traversal strategy is split in two, where two different embedding methods are used; Perch and AudioMAE. The random selection strategy is run 5 times per seed.

5.4 Generalization

The active learning methods that have been tested show promising results. Although the querying strategies have shown to be difficult to fine tune, many of the methods show a significant improvement over the baseline. As the active learning has so far only been executed on the data set $\mathcal{D}_{0.2}^0$, a bias could exist towards this data set. For example, the *top 10 entropy* might be best in this data set due to the lengths of the audio sources, or some other unknown reason. The results might only hold for $SNR = 0$ or $r = 0.2$, as this is all that has been tested. It is therefore important to test active learning in a slightly different domain. To evaluate the generalization of the models, and active learning in general, these models are tested on other data sets. This is to evaluate how the underlying data set structure affects the impact of active learning.

5.4.1 $\bar{\mathcal{D}}_{0.2}^0$ (Domain Change)

Changing the domain, by using sound sources with slightly different origins, gives an insight to how well the model generalises to new type of sounds. This is important as there exists a vast diversity of sounds in this field of study. The data set $\bar{\mathcal{D}}_{0.2}^0$ is used to perform these tests, see section 3.2.2. The underlying distribution of event lengths and the sound sources differs from the previous data set $\mathcal{D}_{0.2}^0$. The same methodology as always is used in this case where 80% of the data in $\bar{\mathcal{D}}_{0.2}^0$ is used for training, whilst the remaining 20% is kept for validation.

Figure 5.16 shows the total IoU performance of using *top 10 entropy* and *mean entropy* as querying strategies, along with the baseline strategy, on the new data set $\bar{\mathcal{D}}_{0.2}^0$. The results for the same metric on the data set $\mathcal{D}_{0.2}^0$ is plotted beside it for comparison. It is apparent that the models and active learning frameworks adapt and generalise well to the new data. Both querying strategies performs better than the baseline, where the performance of *top 10 entropy* is significantly better. The general performance is better for all strategies, including the baseline, in this new data set. This shows that active learning works well, even though the data is somewhat easier.

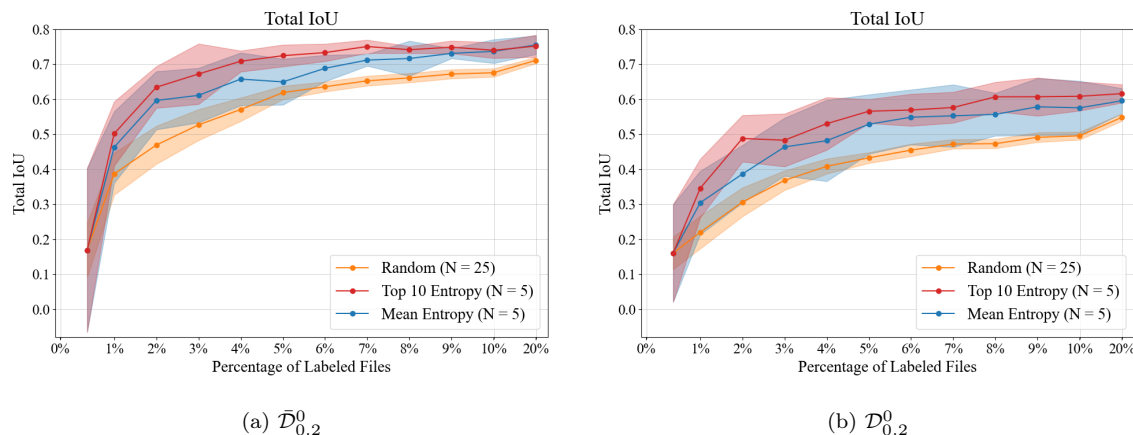


Figure 5.16: **Total IoU: Domain Change** - The total IoU results averaged over 5 seeds ($N=5$), using the querying strategy *top X entropy* and *mean entropy* for the data sets (a) $\bar{\mathcal{D}}_{0.2}^0$ and (b) $\mathcal{D}_{0.2}^0$. The data sets are generated from different domains. All strategies are displayed with 95% confidence intervals. The results shown goes up to 20% of the data set. The baseline is run 5 times per seed.

5.4.2 $\mathcal{D}_{0.2}^{\pm 10}$ (SNR Change)

Trying out different values of *SNR* allows for evaluation of how sensitive the models are to power differences between the noise and the events. This is a problem in real life data as the sound sources (animals) rarely has a fixed distance to the source of which it is recorded. The amplitude of the sounds (animal vocalizations or noise) can also differ, depending on its source. This makes the power of the same types of event differ. Thus, this test is reasonable to make in this field of study, as we can expect the *SNR* to change within data sets and between data sets.

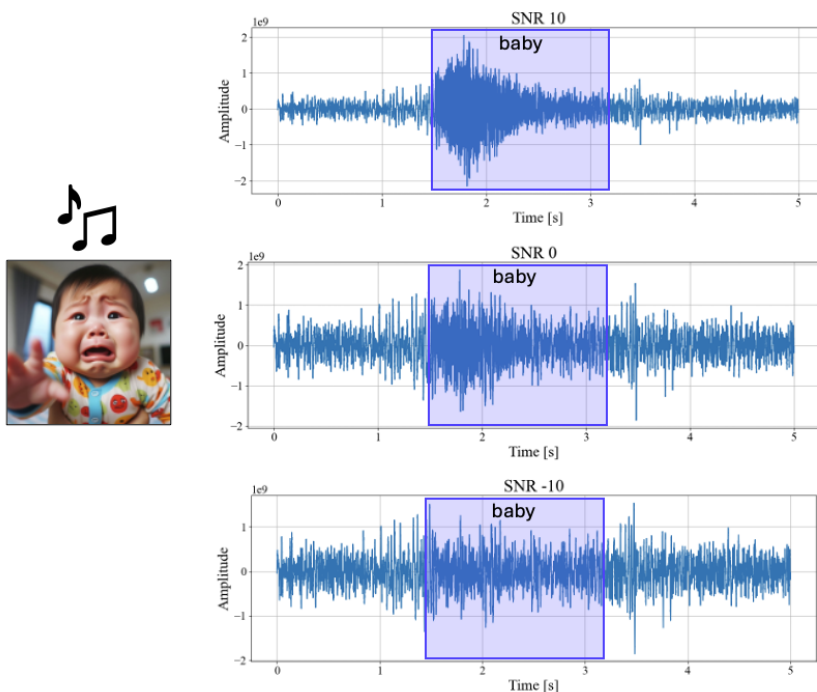


Figure 5.17: **Example: SNR Comparison** - A visual comparison of *SNR* using the waveform of an audio file. The three *SNR*-values plotted are 10, 0 and -10.

Two different values of SNR are studied, 10 and -10. Having a high, positive value of SNR leads to events in the audio files being more distinguishable from the noise. The opposite goes for negative values, where the noise dominates the events in power, making it difficult to even hear the events. This concept is easier to comprehend if one listens to the same recording with different values of SNR , but the difference can also be visualised in the audio file’s waveform as seen in figure 5.17.

In figure 5.18, the performance of *top 10 entropy*, *mean entropy* and baseline is presented for SNR -10, 0 and 10. Note that only the first 12 iterations are plotted. We can see that performance is higher for the data with larger SNR , and this can be observed across all strategies. This is expected as events are now more distinguishable. Besides this, we see that the aggregation strategies outperforms the baseline significantly in the early stages for $SNR = -10$. As noise becomes more dominant it is evident that querying files containing events are of great importance. As no other significant changes in relation to the baseline can be observed, we can establish that active learning seems to be overall robust. It generalises well to changes in the power of noise and outliers. This is a good result in this field as real recordings from nature might contain strong winds, storms and other outliers, and thus the SNR might fluctuate.

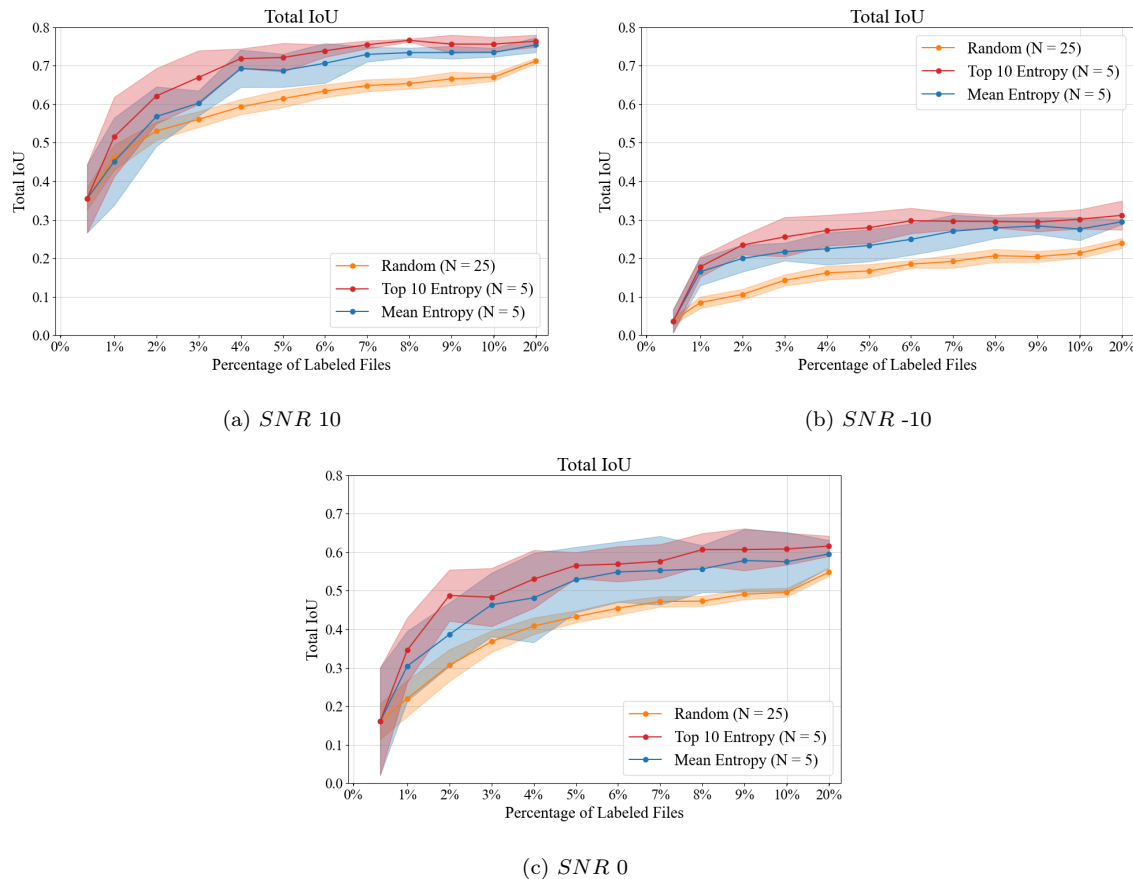


Figure 5.18: **Total IoU: SNR Change** - The total IoU results averaged over 5 seeds ($N=5$), using the querying strategy *top X entropy* and *mean entropy* for data sets (a) $\mathcal{D}_{0.2}^{10}$, (b) $\mathcal{D}_{0.2}^{-10}$ and (c) $\mathcal{D}_{0.2}^0$. The SNR is changed between the data sets. All strategies are displayed with 95% confidence intervals. The results shown goes up to 20% of the data set. The baseline is run 5 times per seed.

5.4.3 $\mathcal{D}_{1.0}^0$ (Event Ratio Change)

We have previously established that a major class imbalance is present in our current data set $\mathcal{D}_{0.2}^0$. Not only between the event classes themselves, but also between events and noise. This is to reflect the real life data that exists in this field, where events might be rare. Due to this, it is important to evaluate the impact of the event to noise ratio, and how active learning performs when the ratio of event segments compared to noise segments increases.

Thus, we test on a data set consisting solely of files containing events, $\mathcal{D}_{1.0}^0$. Except for the ratio of event files, all other generation parameters are kept the same. Figure 5.19 shows the total IoU performance when using *top 10 entropy* and *mean entropy* as querying strategies, along with the baseline, on the new data set $\mathcal{D}_{1.0}^0$. Note that only the first 12 iterations are plotted to emphasise the crucial part of active learning. The results for the same metric on the data set $\mathcal{D}_{0.2}^0$ is plotted beside it for comparison. We see that the *top 10 entropy* strategy still is advantageous compared to the baseline, whereas *mean entropy* performs slightly worse. The baseline appears to perform better on the data set with more events. This can be explained by the fact the entire data set now contains files with events. The baseline’s random querying will thus select files containing events at each iteration, and perform better as the model is exposed to more event data. For the data set with fewer events, the strategies were rewarded for finding events, whereas now it is likely that increases in performance stem from finding more events per file, or files with higher variability between events. Considering that the *mean entropy* does not outperform the baseline, it is possible that this strategy is better suited for the data set with fewer events. This shows once again that the *top X entropy* has great potential, as it appears to be more robust to changes in event frequency.

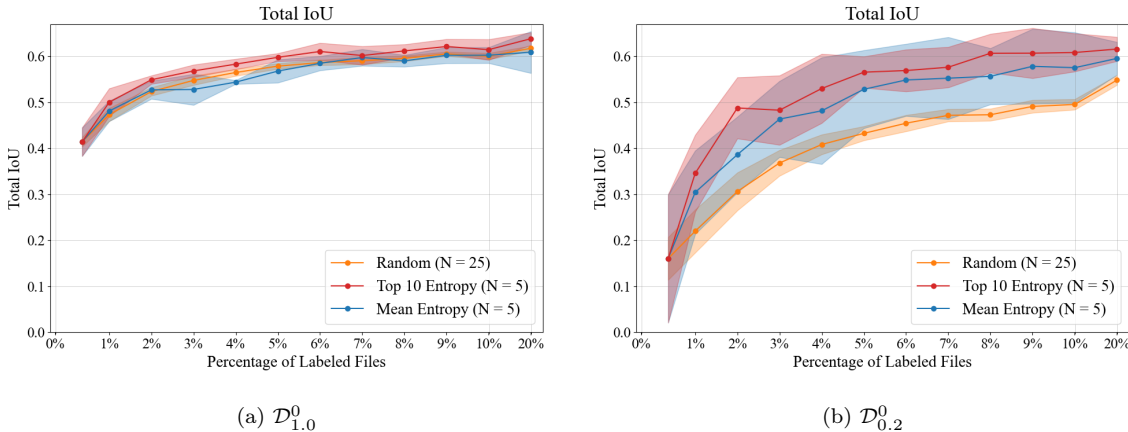


Figure 5.19: **Total IoU: Event Ratio Change** - The total IoU results averaged over 5 seeds ($N=5$), using the querying strategy *top X entropy* and *mean entropy* for the data set (a) $\mathcal{D}_{1.0}^0$ and (b) $\mathcal{D}_{0.2}^0$. All strategies are displayed with 95% confidence intervals. The results shown goes up to 20% of the data set. The baseline is run 5 times per seed.

In figure 5.20, the performance for the individual classes is shown on the $\mathcal{D}_{1.0}^0$ data set. This shows one interesting result, which is that the baseline appears to outperform the active strategies when considering total IoU for meerkat events. This emphasises one of the struggles experienced in this thesis. In general, we have seen that meerkat events have been underrepresented due to being shorter in length. This has lead to our aggregation strategies neglecting them, except when they occur in files containing other events. The baseline model on the other hand have had a struggle with meerkat classification, and other classes, due to its randomness in querying. But now when the baseline model queries on a data set consisting solely of events and is thus presented with more meerkat events, it outperforms the active learning strategies. In conclusion, active learning seem to have a larger impact the more noise being present in the data. This makes sense as the assignment of querying files containing events gets more crucial with fewer events.

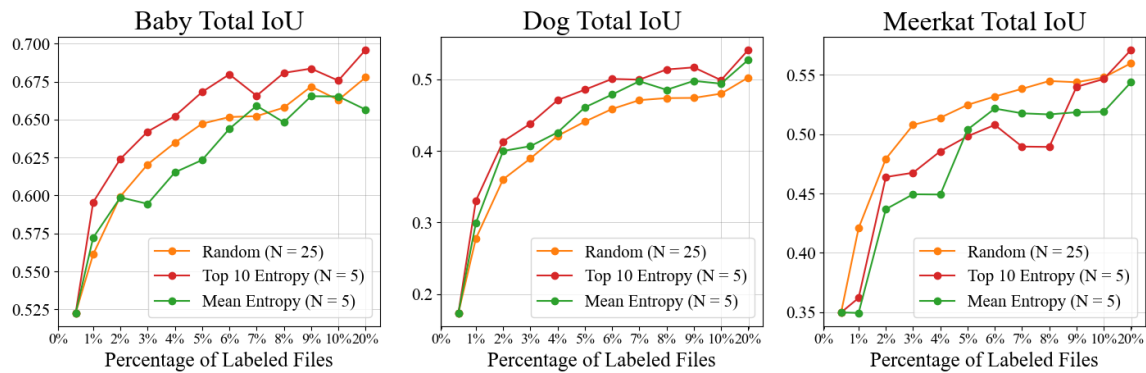


Figure 5.20: **Total IoU on Classes: Event Ratio Change** - The total IoU performance on each individual class averaged over 5 seeds ($N=5$), using the querying strategy *top X entropy* and *mean entropy*, along with the baseline model, for the data set $\mathcal{D}_{1.0}^0$. The results shown goes up to 20% of the data set. The baseline is run 5 times per seed.

Chapter 6

Conclusions

In this chapter, we present the conclusions of this thesis.

The annotation cost for auditory data is a serious issue for monitoring and quantifying biodiversity. We have proven that active learning can be successfully used to drastically reduce the amount of annotated data needed for a bioacoustic SED model.

In the initial testing of active learning, it is shown that for classification, measured in recall, the demand for data is reduced by up to 97% as compared to the baseline. When looking at the SED performance, as measured by total IoU, the reduction in data is around 92%. The active learning strategy to accomplish these reductions is *top X entropy*. Also the aggregation strategy *mean entropy* performs well, where both are proved to be significantly better than the baseline. These results are in line with previous works, in terms of reductions in amount of data needed [5].

For the main data set used ($\mathcal{D}_{0.2}^0$), results point to *top X entropy*, $X=10$, being the most successful entropy option. It consequently and significantly outperforms *mean entropy* on this data set with regards to SED performance. This result is likely due to *top X entropy*, $X=10$, being able to perform well at both noise and events, where as for *mean entropy*, there appears to be a trade-off between event and noise performance.

In general, the active learning performance generalises very well, indicating that it can be useful in many different scenarios. Across all data sets that were tested, the best performing active learning strategy outperformed the baseline. SNR and changes in data distribution or audio recordings does not seem to have any large impacts on how useful active learning is. However, we can conclude that the ratio of files with events in the data set is closely tied to the benefit of active learning. We can conclude that a higher concentration of event files gives a smaller improvement. This makes sense as active learning becomes less about finding files with events, and more about finding many events within files, or finding valuable events that increase performance.

Fine-tuning the active learning frameworks proved to be a difficult task. Diversification attempts and tuning of the X parameter in *top X entropy* showed small improvements at best, that were not statistically significant. This does not mean that they don't have potential to improve the active learning. For the data sets used, the benefit of active learning used is rooted in finding a lot of events rather than diversity. This does not have to be the case for other data sets. There could also exist flaws in the diversification methodology, as it seems that the embeddings used for diversification better represents the noise in the data rather than the events. Maybe a better representation of the contents in the files could lead to better diversification results.

With the *top X entropy*, no optimal X was found. The proposed *top X ensemble* appears to be at least as good as a fixed *top X entropy* strategy, meaning it could be a good way of applying the active learning methodology on a completely new domain with a known distribution, as it adapts to the underlying distribution. It should however be noted that one might be able to do this with a simpler strategy, for instance by using a fixed X that is derived from the distribution. Further, it is important

to consider the value of X with relation to the total number of segments in the file when applying this method to a new problem, as this could have an impact. An interesting prospect for future research is to study an X which is dynamic, and varies with the amount of labeled data L_r .

One observation that was made throughout experiments is that the class with shortest events (meerkats) were not only underrepresented in terms of available training data, but also in active queries, further exaggerating the class imbalance in the labelled data set. It is difficult to conclude if this is an inherent trait of this methodology, or if this is a result of the model being more certain in its predictions of the shorter meerkat class.

The results show that active learning is worthwhile in bioacoustics. The framework in this thesis could be adapted in real life, as it is designed to provide an annotator with reasonable 10 seconds of audio at a time for labelling. With the successful aggregation strategies proposed in this thesis, annotators won't have to struggle to classify 0.12 second audio segments without context. The annotation cost has potential of being drastically reduced. In this thesis, we have only studied annotation cost as the number of audio files, assuming they are equally expensive to annotate. In order to compute the annotation cost in terms of time or money, one would have to consider more precisely how expensive files are to label and if there are any differences in annotation cost between files, something we encourage other researchers to continue with.

With climate change being on the rise we are in desperate need of technological advancements that can help protect ecosystems and biodiversity. Advancements in AI and machine learning are impressive in their own right, but might fail to make a real impact due to monetary limitations or data restrictions. In this thesis, we have hopefully narrowed the gap between cutting-edge technology and applications that help the planet.



Figure 6.1: A baby, a dog and a couple of meerkats in a park.

Bibliography

- [1] Dan Stowell. Computational bioacoustics with deep learning: a review and roadmap. *PeerJ*, (2017):1–32, 2022.
- [2] Ella Browning, Rory Gibb, Paul Glover-Kapfer, and Kate Jones. Passive acoustic monitoring in ecology and conservation. Technical report, 10 2017.
- [3] Annamaria Mesaros, Toni Heittola, Tuomas Virtanen, and Mark D. Plumbley. Sound Event Detection: A tutorial. *IEEE Signal Processing Magazine*, 38(5):67–83, 2021.
- [4] Burr Settles. Active Learning Literature Survey. (January), 2009.
- [5] Zhao Shuyang, Toni Heittola, and Tuomas Virtanen. Active Learning for Sound Event Detection. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 28:2895–2905, 2020.
- [6] Yu Wang, Mark Cartwright, and Juan Pablo Bello. Active few-shot learning for sound event detection. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2022-September:1551–1555, 2022.
- [7] Clemens-Alexander Brust, Christoph Käding, and Joachim Denzler. Active learning for deep object detection. *CoRR*, abs/1809.09875, 2018.
- [8] Asma Yamani, Albandari Alyami, Hamzah Luqman, Bernard Ghanem, and Silvio Giancola. Active learning for single-stage object detection in uav images. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1849–1858, 2024.
- [9] Andreas Kirsch, Sebastian Farquhar, and Yarin Gal. A simple baseline for batch active learning with stochastic acquisition functions. *CoRR*, abs/2106.12059, 2021.
- [10] Gui Citovsky, Giulia DeSalvo, Claudio Gentile, Lazaros Karydas, Anand Rajagopalan, Afshin Rostamizadeh, and Sanjiv Kumar. Batch active learning at scale. *CoRR*, abs/2107.14263, 2021.
- [11] Aymane Abdali, Vincent Gripon, Lucas Drumetz, and Bartosz Boguslawski. Active learning for efficient few-shot classification. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023.
- [12] Ajay J. Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2372–2379, 2009.
- [13] John M. van Osta, Brad Dreis, Ed Meyer, Laura F. Grogan, and J. Guy Castley. An active learning framework and assessment of inter-annotator agreement facilitate automated recogniser development for vocalisations of a rare species, the southern black-throated finch (*poephila cincta cincta*). *Ecological Informatics*, 77:102233, 2023.
- [14] Ines Nolasco, Shubhr Singh, Veronica Morfi, Vincent Lostanlen, Ariana Strandburg-Peshkin, Ester Vidaña-Vila, Lisa Gill, Hanna Pamuła, Helen Whitehead, Ivan Kiskin, Frants H. Jensen, Joe Morford, Michael G. Emmerson, Elisabetta Versace, Emily Grout, Haohe Liu, Burooj Ghani, and Dan Stowell. Learning to detect an animal sound from five examples. *Ecological Informatics*, 77(May), 2023.

- [15] Burooj Ghani, Tom Denton, Stefan Kahl, and Holger Klinck. Global birdsong embeddings enable superior transfer learning for bioacoustic classification. *Scientific Reports*, 13(1), December 2023.
- [16] Mattias Ohlsson and Patrik Edén. *Introduction to Artificial Neural Networks and Deep Learning*. 1 edition, 2022.
- [17] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.
- [18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.
- [19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.
- [20] Google. Bird vocalization classifier. <https://www.kaggle.com/models/google/bird-vocalization-classifier/tensorFlow2/bird-vocalization-classifier/4?tfhub-redirect=true>, 2021. Accessed: 2024-05-09.
- [21] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021.
- [22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [23] Po-Yao (Bernie) Huang, Hu Xu, Juncheng Billy Li, Alexei Baevski, Michael Auli, Wojciech Galuba, Florian Metze, and Christoph Feichtenhofer. Masked autoencoders that listen. *ArXiv*, abs/2207.06405, 2022.
- [24] Ines Nolasco, Shubhr Singh, Veronica Morfi, Vincent Lostanlen, Ariana Strandburg-Peshkin, Ester Vidaña-Vila, Lisa Gill, Hanna Pamuła, Helen Whitehead, Ivan Kiskin, Frants H. Jensen, Joe Morford, Michael G. Emmerson, Elisabetta Versace, Emily Grout, Haohe Liu, Burooj Ghani, and Dan Stowell. Learning to detect an animal sound from five examples. *Ecological Informatics*, 77:102258, November 2023.
- [25] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.
- [26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.