Adapting object detection techniques for beat tracking in musical audio

음악 오디오의 박자 추출을 위한 각체탐지 기법 적용

2023년 1월

서강대학교 대학원
아 트 & 테 크 놀 로 지 학 과
안 재 훈

Adapting object detection techniques for beat tracking in musical audio

음악 오디오의 박자 추출을 위한 각체탐지 기법 적용

지도교수 정 문 열

이 논문을 미디어공학 석사 학위논문으로 제출함

2022년 12월 30일

서강대학교 대학원
아 트 & 테 크 놀 로 지 학 과
안 재 훈

논 문 인 준 서

안재훈의 미디어공학석사 학위논문을 인준함

2023년 1월 3일

주심 정 다 샘 ㊞

부심 정 문 열 ㊞

부심 김 주 섭 ㊞

Acknowledgements

I would like to acknowledge the following individuals:

My professor, Moon-Ryul Jung, for not only providing feedback but also helping me debug code and figure out solutions to problems along the way even during weekends, during what would be a very boring and lonely process. With his guidance, I learned so much during this research project.

Hyukhun Koh, a former intern at the lab who really made me look like the intern, for working together during the earlier stages of this project. His participation always excited and motivated me to work even harder.

Kirak Kim, a fellow graduate student. His consistent participation to the weekly seminars and sharing of opinions as I presented the weekly progress was very beneficial.

Christian Steinmetz, the main author of the WaveBeat paper, for his openness and willingness to help, answering many of the questions I asked regarding WaveBeat over email.

Lastly, my friends and family, for their continuous encouragements, prayers, and support.

Abstract

Beat and downbeat tracking is a fundamental task in music signal processing and has numerous applications in music analysis and generation, but most neural networks rely on the use of dynamic Bayesian networks (DBNs) to extract a final set of predicted beats. This paper proposes a DBN-free approach for beat and downbeat tracking while leveraging components commonly found in object detection. The FCOS object detection model is modified to take 1D music audio as input using the WaveBeat beat tracking model as its backbone, integrated with a Feature Pyramid Network (FPN) to capture hierarchical features. One key contribution is the elimination of the need for post-processing with DBNs, which are traditionally used for external correction. This is done by teaching the model to produce a final set of beats by learning not just beat times but also the length of intervals between each pair of beats. This approach is compared with the same music datasets used by WaveBeat, showing competitive results when matched up with WaveBeat using DBNs. This work also demonstrates the ability of using object detection techniques for beat and downbeat tracking in musical audio without significant changes.

Keywords: beat tracking, downbeat tracking, music information retrieval, object detection, neural networks, deep learning, artificial intelligence

Contents

List of Figures

List of Tables

1 Introduction

Music information retrieval (MIR) is a field related to the science dedicated to the development of methods and systems for extraction of concrete metadata from music. Tasks within this interdisciplinary science are diverse, ranging from genre recognition, key detection, mood, lyric transcription, beat and downbeat tracking, and more. Although research in the field of MIR is noticeably smaller in comparison to those of other disciplines, it is growing in relevance and importance with the growth of machine learning, music streaming platforms, virtual metaverse worlds, and more.

Among these areas of research in MIR involves the task of beat and downbeat tracking, which involves the prediction and estimation of audible beat and downbeat positions in music audio. Generally, beat tracking is a simple and common task done by humans without much thought or effort, usually performed by the tapping of the foot in sync with the music they are listening to. Due to the comparative ease humans have over computers in performing this task, it is often a surprise to many that research involving this task to be performed by computer programs has been undergone for decades yet still suffer from many shortcomings. Although the importance of beat tracking remains prevalent and even continues to grow more relevant as time progresses, there continues to be an absence of beat tracking systems that can reliably predict beats and downbeats when provided with a diverse set of music. Because

music is a part of virtually every culture but can vary immensely across the world, beat tracking systems must be blind to culture, adaptable to diverse genres of music, and able to make meaningful predictions despite complex and uncommon time signatures in order to be considered reliable.

As the world becomes more digital and technology becomes a more critical and unavoidable aspect of daily life for humans across the world, a rise of importance and demand for research in MIR has also taken place. There is a clear demand for the development of a reliable beat and downbeat tracking model. It is difficult to argue against the ubiquity and influence of music throughout even the most different of cultures and its influence on society (Roy and Dowd 2010). Coupled with this fact, such a system would be a godsend to human DJs, if deemed reliable enough, to remix audio or to transition into new music, lined up by beats in order to reduce friction during this process. Along with other artificial intelligence-based innovations in regards to music, non-human DJs can and will become a reality, and the ability to track beats in music would be fundamental to such an application. Such a system can also help in real-time light effects in music performances, concerts, and music festivals. For video editors, mundane and time-consuming tasks based on the beat or tempo of the underlying music can be automated, particularly in which scene transitions occur in sync with, and applying special effects with colors and lighting in a way that corresponds to the mood of, the underlying music. With increased interest in the metaverse in recent years, the demand for improved beat tracking has also grown. Such a beat tracking model can also be used by avatars in virtual environments, allowing them to dynamically perform dance moves when provided beat and downbeat positions of music.

However, it is not as easy for computers to track beats and downbeats as it is for humans. As the concept of beats is extremely basic even to young children, it is difficult at first to imagine that such a simple and natural task is not very straightforward for a computer to be taught to do. Early approaches to beat tracking relied on onset detection to decipher beat positions. However, this approach is imperfect as beats can be present even when no particular sound is being played. It has also been known for a relatively longer period of time that the perception of a beat and its concept is not innate to humans. Experiments involving infants and toddlers have shown that, when given a metronome to synchronize to, only 25% of three year-old children succeeded while 96% of six year olds succeeded (Hargreaves et al. 1986), demonstrating that beat tracking by humans is a learned skill and not intrinsic to humans from birth. This is possibly a reason why all modern research involving beat tracking has involved machine learning techniques to some extent (see Section 1.3), which follows the philosophy in which a computer should learn how to distinguish beats as a human learns how to do so during early stages of development.

1.1 Directions

Currently, there appears to be multiple directions of research in regards to beat tracking and the improvement of model performance. Among these methods include the production of more labeled data so that the model has more diverse data to learn from, improvement of present supervised learning-based models, creation of data augmentation techniques to raise the diversity of the same dataset, usage of self-training to accommodate for a relative shortage labeled data, and implementation of self-supervised learning such that patterns in music can be learned even without the presence of labels.

More labeled data Music beat annotation data is not plentiful, and downbeat annotation data is even more limited in quantity. Recent beat tracking neural networks (Heydari et al. 2021)(MatthewDavies and Böck 2019)(Steinmetz and Reiss 2021) have used a combined total of about nine or so datasets, which includes Ballroom (Gouyon et al. 2006)(Krebs et al. 2013), Beatles (Davies et al. 2009)(Harte 2010), Hainsworth (Hainsworth and Macleod 2004), Carnatic (Srinivasamurthy and Serra 2014)(Srinivasamurthy et al. 2015), RWC Popular (Goto et al. 2002), GTZAN (Tzanetakis and Cook 2002)(Marchand and Peeters 2015), Rock Corpus (De Clercq and Temperley 2011), SMC (Holzapfel et al. 2012), and Simac (Gouyon 2006), with the latter three datasets lacking downbeat data entirely. Even though this exceeds 60 hours of beat-labeled audio, it would be ideal for models to have hundreds if not thousands of hours of music to be able to learn from. Creating more labeled data would lead to models having more data to learn beats and downbeats from, and such data would ideally be diverse in genre, tempo, and time signatures, allowing the model to have more confidence in predicting despite culture and genre. However, listening to music and adding labels to the approximate time points in which a beat is present is not only difficult but also time-consuming, and thus very expensive to create.

Improvement of model Although new datasets continue to be created to mitigate the issue regarding lack of beat data, more research has been spent towards improving the efficiency of existing beat tracking models such that more features can be learned from existing labeled data. A key development in the introduction of neural networks to the research of beat tracking was the usage of recurrent neural networks (RNNs) and long short-term memory

(LSTM) networks which provided support for temporal dependencies, leading to breakthroughs in performance when compared to onset detection-based beat tracking systems (Böck and Schedl 2011). Another key development was the introduction of temporal convolutional networks (TCNs), which refer to a sequence of heavily dilated convolutional layers that provide a large temporal context to the network, which was first popularized in audio waveform generation (Oord et al. 2016) prior to its usage in beat tracking. The use cases for TCNs and RNNs largely overlap as they both deal with temporal-based problems, but studies demonstrate the general superiority of TCNs in training efficiency as well as performance (Gopali et al. 2021), eventually becoming the most dominant approach in more recent research, and were eventually introduced to beat tracking models as well (MatthewDavies and Böck 2019). More recently has been the use of Transformers, a type of neural network architecture that learns to weight important aspects of the input data, usually sequence-based data such as audio (Vaswani et al. 2017). The Transformer architecture has been implemented for beat tracking in (Zhao et al. 2022)(Hung et al. 2022), with (Hung et al. 2022) combining Transformers with TCNs instead of replacing them entirely.

Improvement of post-processing Before neural networks were used to predict beats, the mainstream approach was to use the onsets extracted from an excerpt of music audio to produce a final set of beats using dynamic Bayesian networks (DBNs) ("Bayesian Modelling of Temporal Structure in Musical Audio"). Even after the introduction of neural networks to the field of beat tracking, DBNs continued to be used as a method of producing a final set of beats when provided values from an output activation function of a beat

tracking neural network (Böck et al. 2014). The activation function can be seen as a series of probabilities, each corresponding to a time point in the song, in which a beat is present at that time point. Although a valid method, extracting beats by simply finding the peaks of the activation function outputs (referred to as *peak picking*, see Section 2.3.1) has been found to be unreliable due to its greedy nature and inability to extract a highest globally optimal set of beats. DBNs have continued to receive simplifications and improvements (Holzapfel et al. 2014)(Krebs et al. 2016)(Krebs et al. 2013)(Srinivasamurthy et al. 2015)(Srinivasamurthy et al. 2016) allowing it to work effectively with diverse sets of music genres and styles, from Ballroom dances (Holzapfel et al. 2014) to south Indian music (Srinivasamurthy et al. 2015), and is used in conjunction with practically every modern NN-based beat tracking model. See Section 2.3.2 for more details on DBNs.

Data augmentation Overfitting is a common issue in deep learning in which the model learns too many features specific to the dataset it has been given, memorizing irrelevant aspects of the training data and leading to underperformance when making predictions on unseen data. There are multiple causes for this issue. If training runs for too long, the model can run out of things to learn from the limited dataset and begin to learn too much from it. In the case of beat tracking, a model can overfit the music data when it begins to learn features of the music entirely unrelated to the concept of a beat and not guaranteed at all to be present in other music data. Early approaches involved time stretching and sample rate conversion to simulate slower or faster music (McFee et al. 2015)(Schlüter and Grill 2015), but a more recent approach spectrogram-based models has been to void chances to the raw audio itself but to instead tweak

parameters for spectrogram generation (Böck and Davies 2020)(Hung et al. 2022). Because it is trickier to learn from raw waveform audio data than it is to learn from music represented by spectrograms, much of the focus of research in the production of the non-spectrogram beat tracking model WaveBeat is on data augmentation, in which numerous methods include random white noise, slight adjustments to the beat offsets, pitch shifting, and more (Steinmetz and Reiss 2021). Another recent approach has been to *demix* sounds per instrument type and to have the model learn these patterns separately (Zhao et al. 2022).

Other learning methods Although practically all innovation in deep learning-based beat tracking consisted of supervised learning techniques, other types of learning techniques such as self-supervised learning and semi-supervised learning have gained much interest as potential approaches to improve beat tracking. To accommodate for the lack of labeled music data, self-training can be utilized to produce pseudo-labels, leading to more data that can be passed through the model during training. Although the idea that this approach would actually improve results can seem counterintuitive, usage of self-training has been proven to be considerably effective. As for self-supervised-learning, one factor regarding music data is that labeled data is vastly limited due to the tediousness of the task of label production, unlabeled data, that is, music without beat annotations, is ubiquitous and can be retrieved very easily. Self-supervised learning has not yet been utilized for beat tracking, but can potentially allow for effective pre-training, paving the way for better results when training with actual labeled data takes place. Similar approaches were successfully made in fields such as speech recognition (Schneider et al. 2019), in which unlabeled speech audio data was used in pre-training followed by

training with labeled data. Steinmetz and Reiss make a similar observation in (Steinmetz and Reiss 2021) by stating that utilizing self-supervised learning for beat tracking is a logical next step for innovation in this field.

1.2 Focus

Despite the fact that innovations to approaches involving musical audio beat tracking have been made on a considerably frequent and consistent basis, there is still a long way to go before they can be used in a serious manner. State-of-the-art beat tracking models still struggle in the prediction of beats (especially downbeats) in certain types of music (Böck and Davies 2020)(Steinmetz and Reiss 2021)(Hung et al. 2022)(Zhao et al. 2022). This can be attributed to multiple reasons, and due to the unfortunate lack of labeled data it is even harder to boost performance. Another reason in particular is that the structure of models are not fully optimized to the task at hand, causing in them to underperform in beat tracking even when given the available labeled data.

One interesting observation that we made with the research in beat tracking is that the usage of DBNs for beat and downbeat post-processing was practically never questioned. Although effective in the field of MIR and problems involving sequential data in general, neural networks are powerful and have the ability to learn even unnoticeable patterns in music to process a final set of beats without having to pass it through an external model. In recent development there a single ablation study regarding the use of DBNs was performed in WaveBeat (Steinmetz and Reiss 2021), but did not delve much beyond making a simple comparison with peak picking of the output feature maps for beat and downbeat activation. Even with newer models that use very advanced

techniques (Hung et al. 2022)(Zhao et al. 2022), the usage of DBNs remain almost unquestioned.

The DBN post-processing stage is still used in practically every beat tracking model, and its continued usage has been seemingly unquestioned even by newer and relatively advanced models (Hung et al. 2022)(Zhao et al. 2022). So long as DBNs are used with beat tracking models, the outputs are assumed to not be the actual beats but instead mere observations that can be used to produce a new, better sequence of beat time points. Even as a commonly used type of model within and beyond beat and downbeat tracking, models using DBNs seem to put quite a heavy reliance on the refining process of beats outside the model, especially when a neural network trained on beat and downbeat labels should have the ability to be end-to-end. An ablation test was performed in the creation of WaveBeat by comparing performance using a DBN versus peak picking (Steinmetz and Reiss 2021). Although the improvement with the usage of DBNs was much smaller than in prior spectrogram-based beat tracking neural networks (Böck et al. 2014), DBNs still ultimately prevailed in performance. This motivated us to continue where WaveBeat left off, devising an alternative postprocessing technique that is more effective than peak picking but less complex and more reliant on the model's ability to produce sufficiently accurate beat times, which has the full capacity to predict beats accurately by itself. As a result, we looked towards the field of object detection, in which models are built to learn and detect patterns within feature maps.

In this paper, we proceeded to tackle the problem (and especially the problem of low performance in the detection of downbeats) by changing beat tracking models to learn in a manner that makes DBNs redundant. In the process, we heavily referred to lessons from other problems solved with machine

learning. One fact that must be taken into account is that beat detection and MIR in general is less heavily researched in comparison research in natural language processing and computer vision. During this research, as will be discussed in following sections, we looked to research in computer vision, and applied many of these ideas to the beat tracking problem. Thus, our contributions can be summarized as follows:

- We modified a object detection model, FCOS, to work in beat and downbeat joint detection without having to make significant or fundamental changes. Because research in object detection is very incremental and improvements are frequently made to improve detection results, this approach unlocks a greater repository of lessons that can also be applied to beat tracking.
- We implement a novel loss to resolve a fundamental challenge that is unique in beat tracking and not found anywhere else in object detection, referred to us as the *adjacency constraint* loss. The ultimate goal is to piece together a chain of beats, as they exist as a constant sequence within music and influence the positions of surrounding beats and downbeats. In other words, whether or not there is a beat depends on surrounding beats, but when comparing this problem to object detection, object positions are seldom dependent on neighboring objects, especially those of different classes.
- We add to a backbone of an existing beat tracking model a set of components that act as a substitute for more complex post-processing approaches. To add to this, the final beats are extracted directly from the model without the need to make corrections to the temporal locations. The only post-processing that takes place simply reduces and eliminates

redundancies, or multiple predictions of the same beat or downbeat. This change shifts responsibility for final beat extraction to the neural network from an external component which is limited by its inability to access the intermediate parameters of the neural network.

1.3 Related work

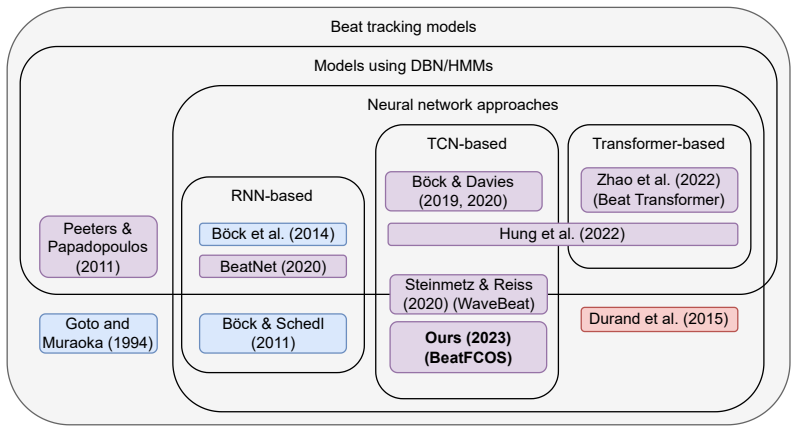

Figure 1.1 Beat tracking models sorted by category. Models marked blue indicate beat-only tracking, models marked red indicate downbeat-only tracking, and models marked purple indicate beat and downbeat joint tracking. Only a select few models were included in this figure based on a number of factors such as relevance, novelty, and time of release.

The first ever well-established beat tracking system was proposed by Goto and Muraoka (Goto and Muraoka 1994), mainly relying on acoustic drum sounds to detect beats. This system was designed for non-MIDI audio, but was limited to popular music with the assumption that all songs have a time signature of 4/4 and that there is no change of tempo, as tempo is often constant in popular music as compared to classical music. Beats are also tracked based on the bass drum, corresponding to the strong beat (first and third quarter

notes in a measure), and the snare drum, corresponding to the weak beat (second and fourth quarter notes).

Böck and Schedl (Böck and Schedl 2011) proposed the first model to rely on artificial neural networks instead of onset detection in the audio, using a bidirectional Long Short-Term Memory (BLSTM) recurrent neural network (RNN) to compare relative differences between sequential spectrograms. This audio signal ultimately gets transformed into an activation function for beats, and an autocorrelation function is also used to detect tempo such that erroneously detected beats can be removed and missing beats can be added. This model was later improved by Böck et al. on two occasions: with the usage of multiple trained models based on music genre and the DBNs for improving the results of the beat activation function (Böck et al. 2014); and changes to the model such that beats and downbeats are jointly modeled (Böck et al. 2016), which followed the first joint beat and downbeat tracking neural network produced by Peeters and Papadopoulos (Peeters and Papadopoulos 2010).

Davies and Böck (MatthewDavies and Böck 2019) proposed an improved model, replacing the BLSTM model from the previous state-of-the-art model by Böck et al. with a temporal convolutional network (TCN), while retaining the spectrogram conversion and DBN post-processing. TCNs have, since their onset, shown to outperform RNN-based models in solving sequential learning problems (Bai et al. 2018) and also resulted in performance improvement in the task of beat tracking. This was later improved to jointly detect not only the downbeat but also the tempo, as these three are strongly related to one another and have proven to help improve performance of each (Böck et al. 2019) (Böck and Davies 2020).

Steinmetz and Reiss (Steinmetz and Reiss 2021) proposed an end-to-end beat tracking model that retains the implementation of TCNs for beat tracking and uses raw audio waveforms instead of spectrograms. To accommodate for the fact that waveforms are more difficult for a neural network to learn than spectrograms, more data augmentation is applied to the raw audio. The removal of the DBN for post-processing was also experimented with, simply passing the features of the TCN blocks' output values through a 1D convolutional layer, where one channel classifies beats and the other channel classifies downbeats, and a simple "peak-picking" method was applied to retrieve timepoints. However, it was found that usage of the DBN after the two channels still produced better results than the peak-picking approach in more scenarios, and thus both sets of scores were reported, one with and without the use of a DBN.

Hung et al. (Hung et al. 2022) proposed a new beat and downbeat joint tracking model that combines the TCN, which has been proven just prior to be effective in beat tracking, with the time-frequency model SpecTNT (Lu et al. 2021), an adaptation of the Transformer model first introduced in natural language processing (Vaswani et al. 2017) used to model both spectral and temporal sequences of time-frequency represented music input with effective results. Shortly after, Zhao et al. (Zhao et al. 2022) also proposed a Transformer-based model referred to as Beat Transformer, differing from the model by Hung et al. by training on demixed audio spectrograms that allow the model to learn the characteristics of each instrument. This structure was based on the notion that general knowledge of the characteristics involving each instrument helps a human beat tracker determine beats and downbeats. Although the model by

Hung et al. model produced slightly greater results, the model by Zhao et al. was demonstrated to be trainable on less powerful hardware.

2 Body

2.1 Music and audio

2.1.1 Music theory

Onset The onset refers to the moment in which a sound begins. The sound in particular can refer to a note or a percussion hit. When listening to a single instrument the onset is relatively easy to determine especially if the audio quality is good and there is little noise, but as more instruments and voices are interwoven this task increases in difficulty. This task can be accomplished by looking for sudden changes to in the amplitude or spectral energy.

Beat The beat can be defined as the basic unit of time in a musical piece, often corresponding with a point in time where a human listener would tap their foot while listening. These beats are also considered to be pulses that correspond to the music, and are crucial as they provide a sense of forward momentum. As to how exactly a moment in time can be determined as a beat is usually quite simple, especially in Western music, as they can be determined by the music's more accented onsets, but this concept can be much more abstract in non-Western music. For example, some music can involve complex polyrhythms, which refers to the simultaneous use of multiple rhythms that do not complement one another, and can sound quite contradictory and non-

harmonious. In such cases, beat tracking becomes very difficult even for human listeners let alone computer systems.

Downbeat The bar (also referred to as a *measure*) is a segment of time consisting of a fixed number of beats, determined by the time signature. The time signature determines the number of beats per bar, and the downbeat is the first beat in the bar. In contrast, the upbeat is the last beat in the previous bar. Compared to other beats in a bar, the downbeat is more often distinguished due to the fact that they determine the rhythm of the music as they define the start of the bar, and is generally more accented in comparison to other beats. Sometimes, they mark the start of a change in dynamic of the song, such as the melody. Thus, the ability to detect downbeats is important for certain tasks such as dancing.

Tempo The tempo simply refers to the speed or pace that a song is played. Sheet music usually has Italian terms referring to the tempo written down at the start of the piece (such as *allegro*), and is commonly quantified in beats per minute (BPM). Tempo can vary heavily across music and can play a direct role in the mood and atmosphere. For example, a slow tempo can provide a sense of tranquility in the music and a fast tempo can provide a sense of action and excitement.

2.1.2 Signal processing

Waveforms An audio waveform (see Figure 2.1) is a graph that visually represents some audio, where the horizontal axis represents time and the vertical axis represents the volume, or amplitude, of the audio signal. These

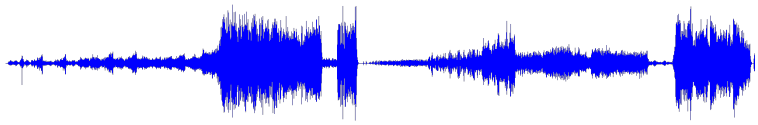

Figure 2.1 An audio waveform representing Ludwig van Beethoven's *Ode to Joy*.

graphs can be displayed by an analog format which represents the waveform continuously, or by a digital format which slices the waveform curve into a list of discrete samples. The most basic waveform is the sine wave, which also produces a very simple sound as it only consists of a single frequency. More complex sounds, such as voices and sound produced from musical instruments, can also be represented by waveforms but the curve would be much more irregularly shaped because it is the result of combining multiple frequencies and harmonics to be represented altogether.

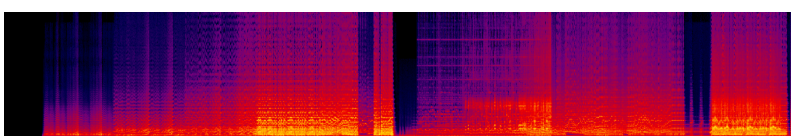

Figure 2.2 A spectrogram representing Ludwig van Beethoven's *Ode to Joy*.

Spectrograms A spectrogram (see Figure 2.2) is a visual representation (usually an image) of a range of signal frequencies with respect to time. Usually, if displayed in 2D format, there are two axes representing time and frequency, and if in three-dimensional format, additionally shows the amplitude of a frequency at a given time. Spectrograms are widely used in audio-related fields of research relating to speech and music, as they are better at showing frequency changes than audio waveforms.

Although much information given some audio can be retained and representated in a series of spectrograms, there is still unavoidable information

loss particularly in regards to the phase. Simply put, once a spectrogram is generated it is not easily convertable back to audio without the use of approximations. An example of a program that attempts to do this is Analysis & Resynthesis Sound Spectrograph (Rouzic 2009).

2.2 Machine learning

2.2.1 Approaches

Supervised learning Supervised learning is an approach to machine learning in which patterns are learned between data and their corresponding labels. Using this approach, the patterns between the data itself and their labels are learned, resulting in a function that is used for the mapping of unlabeled data. In such case, the ideal result would be for the function to properly classify the data despite lack of provided label indicating its class. Most research in beat tracking falls into this category of learning.

Unsupervised learning Unsupervised learning is an approach in which patterns are learned with unlabeled data. The aim is for the model to learn representations given the data provided to it and map out clusters based on similarities and differences among the data.

Self-supervised learning Self-supervised learning can be regarded as an intermediate form of supervised and unsupervised learning, in which pseudo-labels, data in which the labels are automatically given, are used to initialize the model's weights, a process referred to as *pre-training*, in preparation for the actual task to be done with either supervised or unsupervised learning. Pre-training is usually performed using contrastive loss, in which among the data

there is a given target item and the remainder of the data is divided between positive (those that match the target) and negative examples (those that do not match the target). The goal is to match the target with the positive items, minimizing the distances thereof, simultaneously maximizing the distances with the negative items (Tian et al. 2021).

2.2.2 Artificial neural networks

Although computers can generally perform algorithms better, faster, and more efficiently than humans, there are certain tasks that are very easy for humans to carry out but very difficult to a computer. For example, a human can observe a picture and very quickly classify the objects within it, but there is no simple algorithm that can intelligently carry out these tasks. The human brain can be seen less abstractly to be a sophisticated network consisting of hundreds of billions of interconnected neurons that can process information in parallel, and researchers have since successfully implemented systems on the computer consisting of thousands or millions of artificial neurons, loosely modeling human brains for the purpose of bridging this gap (Wang 2003). Referred to as *artificial neural networks* (ANN) and also referred to as *neural networks* (NN) or *neural nets*, they have especially seen many great results in the realm of computer vision and natural language processing. An advantage to using neural networks is that they are built to learn how to perform tasks instead of being built to perform tasks directly. As a result, *a priori* information patterns that have not been noticed or taken into consideration by the programmer can be learned automatically by a neural network.

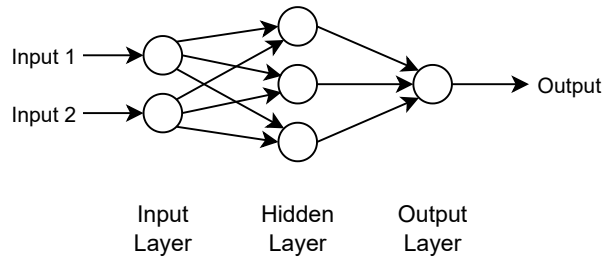

Figure 2.3 An illustration of a simple feedforward network that passes two input values into one hidden layer to provide one output value.

Feedforward neural networks

A feedforward neural network (FNN) (see Figure 2.3), also known as a fully connected network, is a type of artificial neural network widely considered to be the most simple, in which information moves forward from the input nodes to the output nodes. There can also be intermediate nodes that are present, in which case the FNN is considered to be a multilayer perceptron (MLP), and is a single-layer perceptron otherwise. FNNs are non-cyclic, which means that data moves in one direction and can never loop back. When input data is loaded usually in the form of a multidimensional vector, each node in the ANN plays the role of neurons, each producing an output by transforming the input value. The goal for the FNN is for each layer to transform the input data in such a manner where the accuracy of the final result is maximized in a process that is referred to as *learning* (O'Shea and Nash 2015). As FNNs are very general purpose, they are used in a diverse range of tasks.

Convolutional neural networks

A convolutional neural network (CNN) is a type of artificial neural network primarily used to process images and video but is also commonly used for audio

and other data in problems where pattern recognition is necessary. Traditional types of ANNs such as FNNs fall short in the processing of more complex data like images where the number of weights would grow rapidly even with a small increase of size in the input data (for example, a small 64×64 image with three channels for red, green, and blue would result in over 12 thousand weights). In a two-dimensional CNN, the neurons are comprised into three dimensions with the added dimension referring to the depth, or channels corresponding to each position on the 2D area, and the neurons within any given layer of the CNN only connects to a small region of the preceding layer (O'Shea and Nash 2015). CNNs generally comprise three types of layers: convolutional layers, pooling layers, and fully-connected layers.

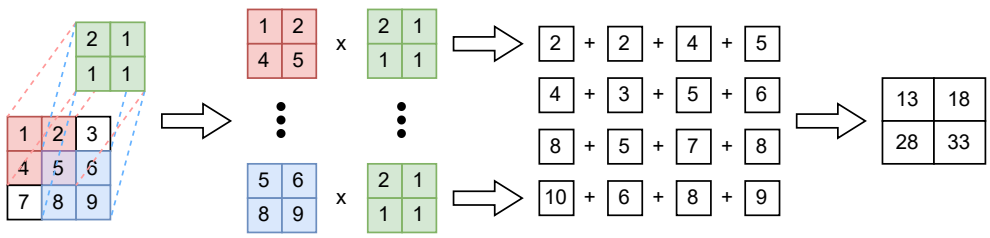

Figure 2.4 An illustration of how a convolution layer processes input data (3×3 in this example) with a filter (2×2 in this example) with a stride of 1, resulting in new output data (which is 2×2 in this example)

A convolutional layer (see Figure 2.4) applies to the data a set of convolutional filters that are designed to extract patterns that are local to the relevant area of the input. These filters generally have a fixed size and stride (distance in which the filter is shifted after each calculation), and consist of a matrix of weights in which element-wise multiplication with the data at the specific location is taken place.

A pooling layer applies downsampling the input data with the goal of retaining as many features as possible. There are several types of pooling

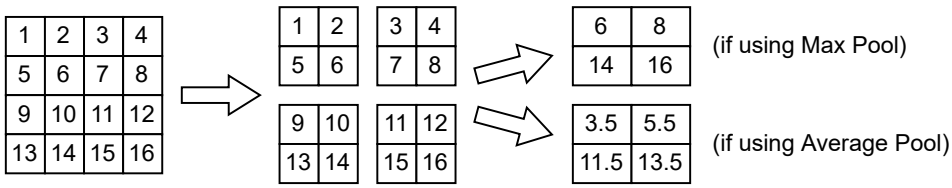

Figure 2.5 An illustration of pooling layers in CNNs, demonstrating both max pooling and average pooling.

operations, but the two most common ones are max pooling (see Figure 2.5), in which the maximum value within the pooling window is chosen, and average pooling, in which the average of all the values within the window is used.

2.3 Beat post-processing

One observation with virtually every modern beat tracking model (Matthew-Davies and Böck 2019)(Steinmetz and Reiss 2021)(Hung et al. 2022)(Zhao et al. 2022) is that some type of post-processing takes place on the output data to retrieve a final set of beats. When the audio is simply passed through a model, a list of probabilities of beats and downbeats at the locations they correspond to, and a final set of beats and downbeats is extracted using a post-processing method. Performing peak picking (see Section 2.3.1) on this set of probabilities is widely considered to be the simplest post-processing method. However, due to inability of peak picking to consider a global optima, DBNs designed to produce the most probabilistic combination of beats are more commonly used in newer models.

2.3.1 Peak picking

In earlier beat tracking models, beat positions were often extracted from the model by choosing the model output, an activation function where probability

of a beat at the corresponding time point. Commonly referred to as *peak picking*, an early version of this approach (Eyben et al. 2010) truncated all activation output values that fall below the activation function output median to 0 and select as beats all time points with activation values whose preceding and succeeding values fall below it. However, this approach falls short in filtering out obvious false positives and negatives that do not align with the general tempo of the song. Smarter approaches were thus implemented, such as tempo estimation, either within the entire song (which assumes constant tempo throughout) or with a shifting window that is able to detect potential changes in tempo after each beat, by finding the most dominant beat distances and to use this information to add missing beats or remove beats that do not align with this overall estimated beat pattern (Böck and Schedl 2011). Although this was proven to be a more effective method, it still had significant problems, especially where the sliding window approach was found to get easily distracted by a few misaligned beat activations, producing a number of poor subsequent beat predictions prior to recovery (Böck et al. 2014).

2.3.2 Dynamic Bayesian networks

Dynamic Bayesian networks (DBN) are a generalized version of hidden Markov models (HMMs), a type of probabilistic graphical model that can be used to represent a set of random variables over time, and is often used for making future predictions based on observations from the past (Murphy 2002). DBNs can be used to find relationships between variables across adjacent time steps and can model hidden states, and are widely used due to their effectiveness in sequential data and representation of hierarchical relationships. It consists of nodes, which represent the random variables, and edges, which represent

the relationships between the variables. The nodes can be divided into two categories: hidden and observable variables. In beat tracking, DBNs can be used to determine the final set of beats and downbeats given an activation map, provided to be used as the observable variable sequence. Usage of DBNs provides the added consideration of musical consistency, or the coherence and logical flow of music, to the set of probabilities that are provided by the beat tracking neural network. When these added considerations are applied to the output of the neural network that is expected to have some level of noisiness, beats and downbeat positions processed by the DBN can be treated as more reliable and accurate. There are two main components of a DBN, the *transition model* which specifies the probability that the hidden variables change from one time point to the next time point, and the *observation model* which specifies the probability that the observable variables will have some values given the values of the hidden variables at all time points.

The particular type of DBN used for music that defines the relationships between the different aspects of music helpful for the inference of beat locations is known as the bar pointer model (BPM). The BPM was first implemented as a DBN for meter analysis (Whiteley et al. 2007), which has been extended and improved (Krebs et al. 2015)(Böck and Schedl 2011)(Holzapfel et al. 2014)(Srinivasamurthy et al. 2015) and is now very commonly used for the post-processing of beats and downbeats.

Hidden variables Each audio frame has a set of hidden variables that provide information about its corresponding bar pointer $\mathbf{x}_k = [\phi_k \ \dot{\phi}_k \ r_k]$. $\phi_k \in [0, M)$ refers to the position of k in the bar with length M of a *bar cycle*. In other words, the bar cycle is divided into M *slots*, each of which can be described

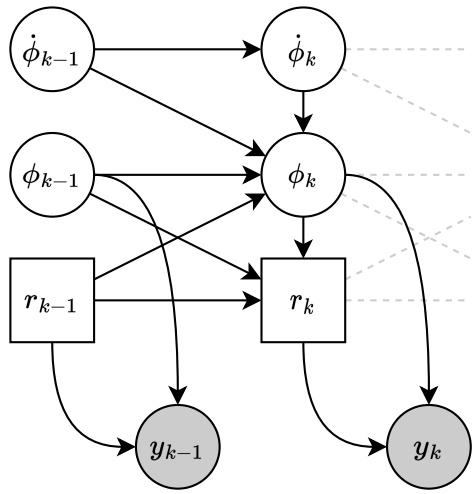

Figure 2.6 A version of the Bar Pointer Model as implemented in (Holzapfel et al. 2014) and (Krebs et al. 2015). White circles and squares are hidden variables and shaded circles indicate the observed variables.

by the bar position ϕ_k. Once ϕ_k reaches M, it is set back to 0 indicating that the value cycles to the start, hence the usage of the word *cycle*. The bar cycle length M is set to 1600 in (Srinivasamurthy et al. 2015). $\dot{\phi}_k \in [\dot{\phi}_{min}, \dot{\phi}_{max}]$ refers to the tempo at k, which is given by the number of bar positions per frame. r_k refers to some feature that would come from a set of collected onsets or a neural network trained prior. The relationship between these values can be seen in Figure 2.6.

Transition model The purpose of the transition model is to map the relationships between variables in the model. The factorized transition model pictured in Figure 2.6 describes these relationships.

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}) = P(\phi_k|\phi_{k-1}, \dot{\phi}_{k-1}, r_{k-1}) \times P(\dot{\phi}_k|\dot{\phi}_{k-1}) \times P(r_k|r_{k-1}, \phi_k, \phi_{k-1}) \tag{2.1}$$

As the BPM shown in Figure 2.6 shows three hidden variables, the factorized transition model $P(\mathbf{x}_k|\mathbf{x}_{k-1})$ has three terms, each corresponding to a single hidden variable. The bar position transition probability is expressed as

$$P(\phi_k|\phi_{k-1}, \dot{\phi}_{k-1}, r_{k-1}) = \mathbb{1}_{\phi} \tag{2.2}$$

in which $\mathbb{1}_{\phi}$ is an indicator function that is 1 if the condition $\phi_k = (\phi_{k-1} + \dot{\phi}_{k-1})\text{mod}M$ is satisfied, meaning that the bar position advances and cycles back to the beginning, and zero otherwise. The tempo transition term is expressed as

$$P(\dot{\phi}_k|\dot{\phi}_{k-1}) \propto \mathcal{N}(\dot{\phi}_{k-1}, \sigma^2_{\dot{\phi}}) \times \mathbb{1}_{\dot{\phi}} \tag{2.3}$$

which is directly proportional to a normal distribution, in which a transition of constant tempo would have highest probability but tempo changes would be allowed but result in lower probability the more abrupt the tempo change is. Lastly, the rhythmic pattern transition probability, $P(r_k|r_{k-1}, \phi_k, \phi_{k-1})$ prevents rhythmic pattern transitions to occur until the end of the bar is reached.

Inference The transition and observation models used to determine where likely beat positions are located are defined as probability distributions, and so after these probability distributions have been established, the goal is to find the sequence $\mathbf{x}^*_{1:T}$ that maximizes the *a posteriori* probability $P(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$.

$$\mathbf{x}^*_{1:T} = \arg\max_{\mathbf{x}_{1:T}} P(\mathbf{x}_{1:T}|\mathbf{y}_{1:T}) \tag{2.4}$$

where $P(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ represents the conditional probability of $\mathbf{x}_{1:T}$ given $\mathbf{y}_{1:T}$, which is defined as

$$P(\mathbf{x}_{1:T}|\mathbf{y}_{1:T}) \propto P(\mathbf{x}_1) \prod_{k=2}^{T} P(\mathbf{x}_k|\mathbf{x}_{k-1}) P(\mathbf{y}_k|\mathbf{x}_k) \tag{2.5}$$

This condition represents the transition and observation models $P(\mathbf{x}_k|\mathbf{x}_{k-1})$ and $P(\mathbf{y}_k|\mathbf{x}_k)$ respectively, with $P(\mathbf{x}_1)$ defined as the *initial state distribution* which can consist of prior known information such as tempo distributions. The state sequence that maximizes the probability of the hidden variables can be computed by using sequential Monte Carlo algorithms that calculate approximations to posterior distributions (Doucet et al. 2001). Another popular method for finding $\mathbf{x}^*_{1:T}$ is the usage of the Viterbi algorithm (Rabiner 1989).

2.4 Methods

2.4.1 Object detection with beats

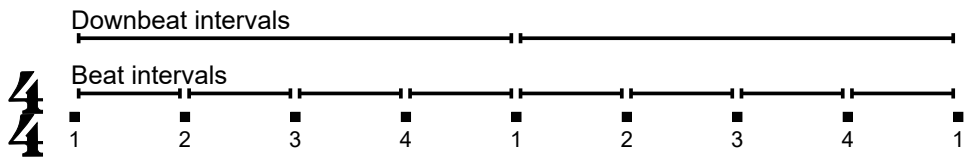

Figure 2.7 Beat and downbeat intervals are generated from a list of beats. Most notably, the downbeat is represented twice: once as a downbeat interval, and once as a regular beat interval.

In order to use object detection models to detect beats, several essential steps were required to take first. The most glaringly obvious step was to reduce the model to work with 1D audio waveform data instead of 2D image data. This was relatively straightforward, as most of this work consisted of changing 2D convolutional layers to 1D layers and adapting 2D algorithms such as IoU or non-maximum suppression to work in a single dimension. The next step involved additional decisions to be made on how beats and downbeats should be

represented. While in object detection the goal is to detect objects represented in 2D in images that compose of 2D, the beat detection task consists of the detection of 0D time-points in 1D audio. This issue could be theoretically resolved by giving beats a fixed length to be represented in 2D or by creating a custom method in which anchors are determined to be positive by replacing the IoU approach with a check to ensure that the point is within the anchor box, but we decided to instead represent beats and downbeats as intervals, with each beat interval endpoints corresponding to two consecutive beats and each downbeat interval corresponding to two consecutive downbeats (see Figure 2.7). The reason for this is that simply representing beats as a 0D point would fail to provide information on the distance between two given beats, which is crucial information for learning when beats appear in music.

2.4.2 Use of WaveBeat as a backbone

Although this work is designed to work with any beat tracking model dealing with both waveforms and spectrograms alike, we decided to use WaveBeat as a starting point due to its well-organized codebase and interesting approach to beat and downbeat tracking.

Temporal convolutional networks

A temporal convolutional network (TCN) is a network used for sequence modeling and is generally distinguished by (1) causal convolutions, in which each resulting timepoint value does not include any information from future timepoints, and (2) the architecture having the ability to map any sequence to an output sequence irrespective of input sequence length, similar to that of an RNN (Bai et al. 2018). The nodes of the TCN used in WaveNet (Oord et al.

2016), a deep neural network used for generation of raw audio waveforms and the neural net that popularized the usage of TCNs, is causal, meaning that it only contains information from the present and previous timepoints of the prior layer and not from any future timepoints. Although the prior definition of TCNs require them to be causal, TCN-based beat tracking models (MatthewDavies and Böck 2019) (Steinmetz and Reiss 2021) generally forego this requirement, effectively using non-causal TCNs, i.e. allowing their nodes' timepoint values to contain information from future timepoints. Davies and Böck explain the reasoning for this directly, indicating that designing the TCN to be causal would be crucial for online (i.e. real-time) beat tracking but decided to allow full access to the extra context as training takes place entirely offline (i.e. training is being done with already existing music beat data).

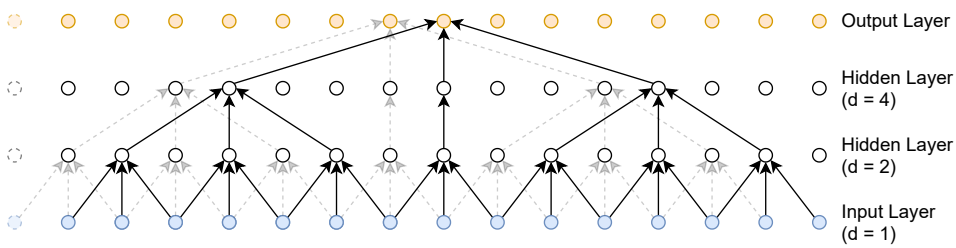

Figure 2.8 An example of a non-causal temporal convolutional network (TCN). The figure above portrays a example, simplified to portray a dilation pattern of (1, 2, 4) and stride of 1 instead of the (1, 8, 64, 512) pattern and stride of 2 found in a single TCN block in WaveBeat.

Dilated convolutions Dilated convolutional layers (see Figure 2.8) support exponential expansion of the receptive field without loss of resolution or coverage by skipping input values by a specified step value. Although pooling or strided convolutions also fulfill a similar purpose, the primary difference of dilation is that the input and output share the same size. This was first

implemented for modern day machine learning by Yu and Koltun in (Yu and Koltun 2015) and was also used for TCNs in WaveNet (Oord et al. 2016). This is useful for storing a wider portion of the audio data in a single node, and for this reason is commonly used in TCNs. In standard convolution, the dilation value is 1. The original intent by Yu and Koltun for the implementation of dilated convolutions was to perform gradual but exponential expansion of the receptive field using a dilation pattern of (1, 1, 2, 4, 8, 16, 1) (Yu and Koltun 2015). WaveNet also applied a similar concept, following the pattern (1, 2, 4, ..., 512). The spectrogram-based TCN model follows the same pattern but goes all the way to 1024 (MatthewDavies and Böck 2019), and WaveBeat instead uses base 8 and has two stacks of (1, 8, 64, 512) (Steinmetz and Reiss 2021). Dilation does not downsample the input data but merely widens the spread of the nodes that would be used to calculate the result node, and increasing the stride is necessary to perform downsampling.

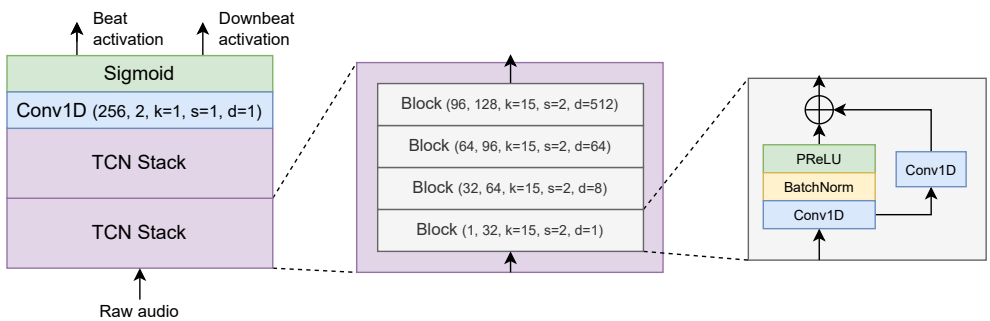

Figure 2.9 A diagram of the WaveBeat architecture.

WaveBeat model architecture

In order to use WaveBeat as a pretrained backbone, all WaveBeat checkpoints used in our experiments were trained using the default hyperparameters, including the configuration of its TCN structure (see Figure 2.9). These will be

reviewed in this section, but we also advise the reader to also refer to their paper (Steinmetz and Reiss 2021) as well as the official WaveBeat repository on GitHub[1].

When a one-channel (mono) audio waveform is provided to the model, it begins to be passed through two TCN stacks, each consisting of four blocks. Each block consists of a 1D convolutional layer, a batch norm layer (Ioffe and Szegedy 2015), and a PReLU activation layer (He et al. 2015) as well as a residual 1D convolutional layer. After passing through the first block the channel rises from 1 to 32 and rises by 32 every time it passes through subsequent blocks, resulting 256 channels after passing through the two stacks. As there are four blocks per stack with two stacks in total, the input data is passed through eight blocks in total. The TCN in WaveBeat differentiates itself from that of the TCN model by Davies and Böck (MatthewDavies and Böck 2019), in that downsampling takes place by setting the stride of each layer to 2, halving the data length after each block, a kernel size of 15, and a dilation pattern of (1, 8, 64, 512) per stack. After the data is passed through the TCN, a 1D convolutional layer with a kernel size of 1 is applied to downmix to 2 channels, one corresponding to beat and the other to downbeats, and a sigmoid layer is applied to produce beat and downbeat activations.

Feature pyramid networks

A feature pyramid network (FPN) is a component commonly used in machine learning tasks where the scale of an object or any other target that is to be detected in a given image largely vary in scale across the dataset. It was

[1] https://github.com/csteinmetz1/wavebeat

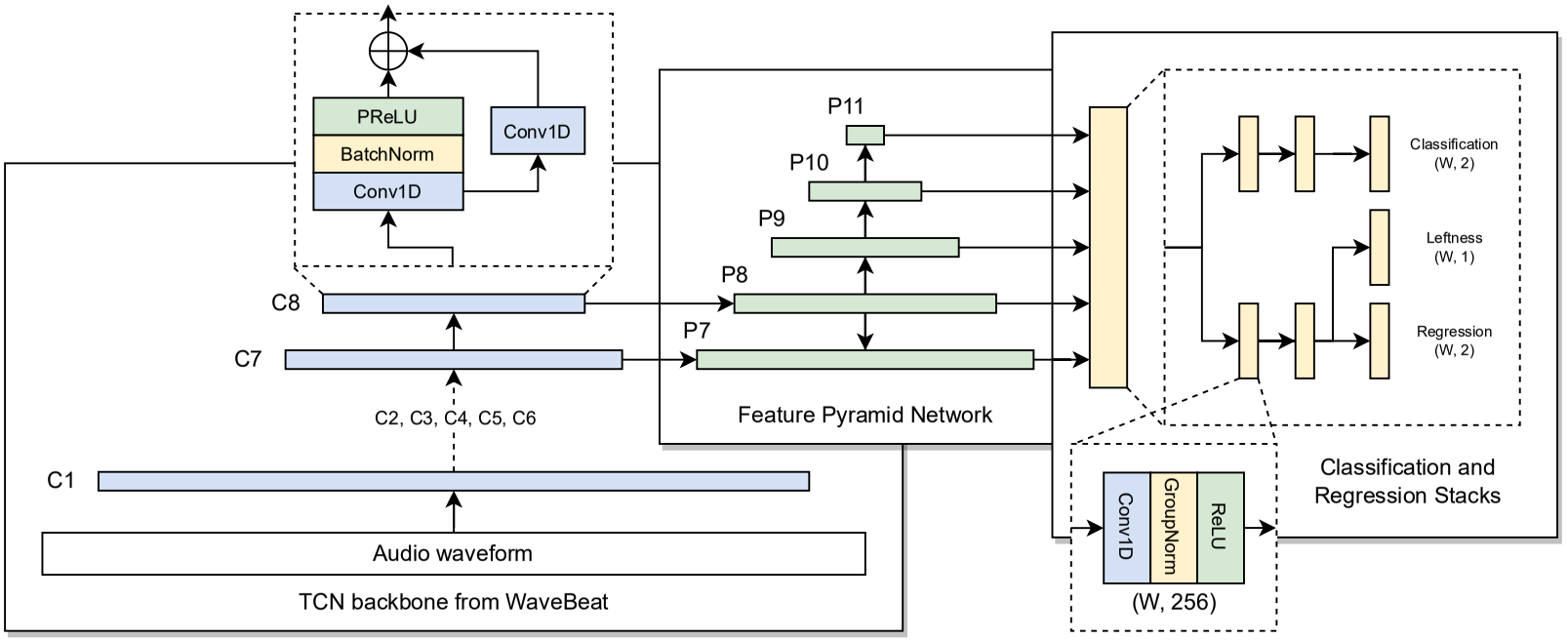

Figure 2.10 Full structure diagram of WaveBeat with BeatFCOS. Instead of the output of C_8 passing through two-channel Conv1D and Sigmoid layers to provide beat and downbeat activations, the C_7 and C_8 outputs are passed to P_7 and P_8, acting as the part that integrates the WaveBeat backbone with our BeatFCOS model.

first implemented for the task of object detection, but was designed to be general-purpose, allowing it to be compatible with segmentation proposal (Lin et al. 2017a) as well. With respect to the handling of scale by the FPN, when applied to object detection it should be able to detect a distant object in the distance as well as it would a nearby object. The philosophy of this is that in certain scenarios it is more useful to refer to a layer that is larger, even though each location on the feature map may have fewer channels (i.e. contain less information about each location on the feature map). In regards to computer vision, this is analogous to the idea of looking at a painting closely to see the smaller details, and backing up further from it to see the larger details. Since many models generate compressed feature maps with more channels per location by passing the input data into intermediary layers, it is an intuitive thought to simply save these intermediate feature maps and pass them into the model. However, the problem with this is that, while the intermediate levels would be larger than the final layer in such model designs, each location of the feature maps usually suffer from lack of information about the location of the input data they represent, as fewer channels are assigned to them. The FPN component was designed to resolve this issue, and it consists of a bottom-up pathway and a top-down pathway.

The bottom-up pathway by itself can also be referred to as the *backbone* of the FPN, the means in which the input data is processed, as the FPN by itself is not designed to directly convert input data to feature maps but to instead work with already existing models. In other words, another model that generates feature maps by scaling down data, substituting scaled down data with more channels per location on the feature map, can be used in conjunction with the FPN. In this part, intermediate feature maps of proportionally decreasing

sizes are produced as the given input data is passed through it. Where most models operate in such a way where the data is passed through multiple layers and the produced single feature map output from just final output layer is exclusively used, the FPN utilizes intermediate layers as well. Thus, a select few intermediate layers are returned with the final layer as well. These layers are then passed together into the top-down pathway.

After the layers are passed into the top-down pathway, the topmost layer is handled in two ways; first, it is passed into a convolutional layer so that the feature size matches and then passed on to be predicted, and second, it is added to the layer beneath (after upsampling it so that elementwise operations can be made possible). This FPN paradigm has continued to enjoy usage in subsequent object detection models, such as RetinaNet (Lin et al. 2017b), FCOS (Tian et al. 2019), VarifocalNet (Zhang et al. 2021) and others.

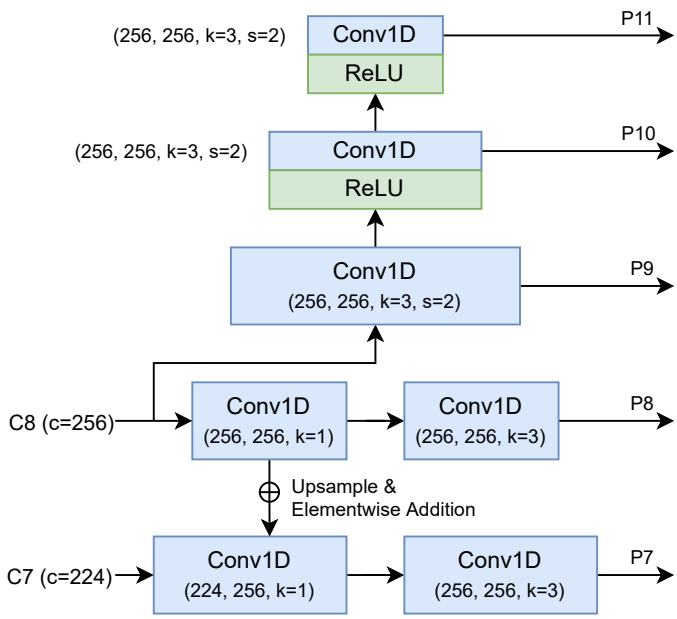

Figure 2.11 FPN used with WaveBeat.

Integration of WaveBeat and FPNs

In order to integrate the WaveBeat backbone with the FPN, the final convolutional and sigmoid layers are removed and the outputs of the last two TCN blocks C_7 and C_8 with 224 and 256 channels respectively are passed into a convolutional layer the last two FPN levels P_7 and P_8. This differs from the implementation of FCOS (Tian et al. 2019) and RetinaNet (Lin et al. 2017b) where the last three blocks from ResNet (He et al. 2016) due to the fact that the memory footprint becomes too large otherwise. We followed the original FPN implementation (Lin et al. 2017a) where the P_8 level is upsampled and added underneath with elementwise addition to add more details to the output of P_7 before both P_7 and P_8 each pass through another pair of convolutional layers. P_9 is created by passing the original result from C_8 into a single convolutional layer. P_9 forms P_{10} by passing its output through ReLU and convolutional layers, and P_{11} is formed from P_{10} using the same way. All convolutional layers in the FPN produce outputs of 256 channels. See Figure 2.11 for more details.

Target base level Especially with the large resolution of the raw input audio (22050 samples per second when using default WaveBeat hyperparameters), creating beat and downbeat intervals on the bottommost level like object detection models causes them to be very wide. As a result, we raised the target base level to the 7th level, the same as the bottommost FPN level P_7.

2.4.3 Anchor points

Anchor points refer to reference points of a feature map, where each is given the responsibility to make class and box boundary predictions. The feature map is usually a downsampled result of the input data, and the model would

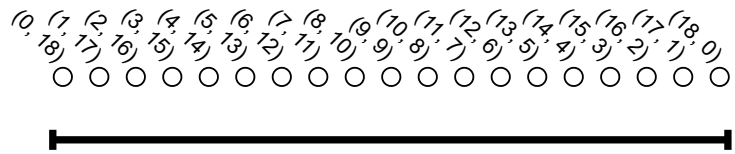

Figure 2.12 A beat interval of length 18 when downsampled to the base level. For the sake of simplicity, each anchor point is spaced apart by a radius of 1 in this example. The coordinate values above the anchor points shows all regression targets.

make predictions from each anchor point on the feature map such as class type as well as the box boundaries of the object it is predicting.

For much of object detection history, approaches involved the shifting of *anchor boxes* (Ren et al. 2015) with predefined aspect ratios and sizes across the features maps, each responsible for determining the class type and box boundaries of an object that may be within itself. During training, most of these models determined whether or not an object was present within its boundaries by checking if the intersection-over-union (IoU) of the anchor box and the object's ground-truth bounding box exceeded a threshold such as 0.5 (Ren et al. 2015)(Lin et al. 2017b). Due to the considerably large number of hyperparameters in such a paradigm, there have been multiple efforts to replace anchor boxes, including CornerNet (Law and Deng 2018) and DenseBox (Huang et al. 2015). The first object detection model to implement anchor points (see Figure 2.12), the DenseBox model suffered from a number of issues, including one in which the many low-quality bounding boxes predicted at each pixel point caused low performance. However, many of these issues were addressed with the release of FCOS (Tian et al. 2019), which can be considered the successor to DenseBox and the first object detection model since YOLOv1 to surpass anchor box models in performance without the use of anchor boxes. The issue

relating to the excess of bounding boxes poor in quality was addressed with the addition of a centerness value used to lower the confidence scores of low quality boxes (see Section 2.4.5).

Positive anchor points

In FCOS (Tian et al. 2019), every anchor point is categorized as either positive or negative. Anchor points are determined to be positive if they are assigned ground-truth bounding box to be predicted, and negative if no such bounding box has been assigned to it. As it can be seen in Section 2.4.5, both positive and negative anchor points are used to train the model in detecting classifications. A positive anchor point would lead the model to learn that there is an object of a class associated with the anchor point, and a negative anchor point would lead the model to learn that there is no object of any class associated with the anchor point. There are multiple factors that determine whether or not an anchor point is positive or not. Among this, the most relevant factor pertains to overlap, in that the anchor point must be within the box. Because FCOS uses FPNs, it also determines the most appropriate level whose anchor points would be most appropriate to be assigned with the ground-truth bounding box by imposing regression limits (see Section 2.4.3). This is to ensure that bigger objects are detected on lower resolution feature maps and smaller objects are detected on higher resolution feature maps. A condition that was added in the second version of FCOS (Tian et al. 2020) is that the anchor must be in a narrow sub-region of the bounding box, located near the center for object detection and located at the left side when applied to beat tracking (see Section 2.4.3). This condition was later added because the best candidates to make predictions on an object are generally located near its center and there was

no point in making all anchor points within an object's bounding box positive because their predictions would be less trustworthy anyway.

Regression limits

In order to properly distribute objects across the levels of the FPN (see Section 2.4.2), FCOS imposed regression limits using base-2 values, in which $m_{i-1} > \max(\mathbf{r}_{\text{left}}, \mathbf{r}_{\text{top}}, \mathbf{r}_{\text{right}}, \mathbf{r}_{\text{bottom}}) < m_i$, where m_2, m_3, m_4, m_5, and m_6 are 0, 64, 128, 256, 512, and ∞ respectively, must be satisfied in order for it to be predicted on that layer (Tian et al. 2019).

Usage of k-means algorithm to determine limits Although many object detection models use simple base-2 values to determine regression limit values and may be sufficient, there can be specific problems in which these values do not apply at all. For example, if all relevant object sizes are unevenly distributed or are disproportionately small or large, performance would not reach full potential. To resolve such an issue, sizes for all the expected data should be considered when determining a fixed number of clusters, which would usually match the number of layers in the pyramid. We used the k-means algorithm, which is an algorithm used to divide a set of samples into k clusters, to help determine the regression limits. In this case, samples refer to bounding boxes irrespective of class type, which are ultimately divided into k clusters $\{b_1, b_2, ..., b_k\}$ based on the bounding box sizes. After using this approach to find the centroids, we calculated the $k-1$ cluster midpoints $\{m_1, m_2, ..., m_{k-1}\}$ where $m_i = b_i + (b_{i+1} - b_i)/2$ and used these as the regression limits for each level of the FPN. The cluster midpoints we used were $\{m_0, m_1, m_2, m_3, m_4, m_5\} = \{0, 0.546, 0.955, 1.588, 2.359, \infty\}$ and were calculated with all of our training data (see Section 2.4.7).

Anchor point sub-box

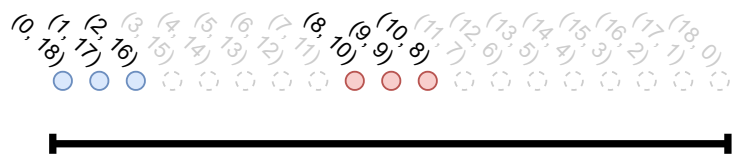

Figure 2.13 A beat interval of length 18 when downsampled to the base level, showing anchor points that overlap. Positive anchors are highlighted blue if the positive anchor point sub-box is at the left, or red if at the center. The coordinate values above the anchor points shows all regression targets. This is the same beat interval as the one shown in Figure 2.12.

Although the original version of FCOS (Tian et al. 2019) determined every location to be positive (i.e. to have responsibility for detecting an object) simply by checking if the location falls into the boundaries of the target bounding box, no matter how far it may be from the center, the second version of FCOS by Tian et al. was updated to limit this condition even further. A hyperparameter r was introduced and this condition was updated such that only location points within the sub-box $(c_x - rs, c_y - rs, c_x + rs, c_y + rs)$ where (c_x, c_y) is the location of the image center and s is the total stride corresponding to the feature map would have the responsibility to predict the object (Tian et al. 2020). The radius r hyperparameter was set to 1.5 in FCOS as they determined it fit for the COCO dataset (Lin et al. 2014), but we learned in our experimentation, this value varies per dataset and testing must take place before deciding on how to set the value. As noted by them, this update update reduced the importance of the centerness loss, but was still kept as it still led to a minor performance improvement, as predicted bounding boxes with low IoU scores but high confidence stores are reduced, and that the computing time added

is negligible (Tian et al. 2020). For 1D interval detection, we implemented a left- and 1D-based version of this radius condition, where the sub-box is defined as $(x_1, x_1 + rs)$. Notably, the original sub-box in FCOS is biradial while ours is not, which led to issues where the sub-box was so small that no anchor points actually landed within it, causing some intervals to end up undetected. We resolved this issue by raising r, but we took advantage of the fact that only two classes exist in the dataset and separated the value by class, with r_{beat} corresponding to the radius for beat intervals and r_{downbeat} for downbeat intervals, setting $r_{\text{beat}} = 2.5$ and $r_{\text{downbeat}} = 4.5$. Because we know that downbeats will never be shorter than beats, allowing downbeat intervals to use a larger sub-box was a rational decision.

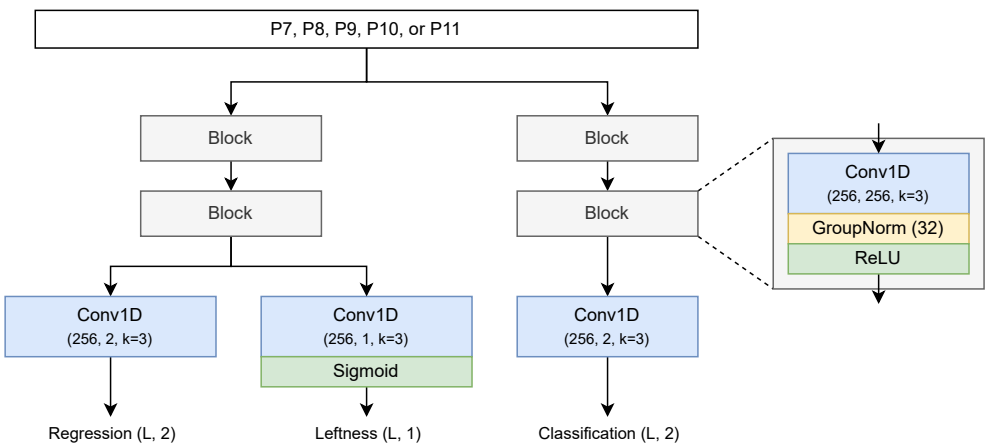

Figure 2.14 The FPN data is passed through a series of convolutional layers before resulting in three prediction values, one for classification, one for regression, and one for leftness scores. Each output from the FPN is passed through these stacks one by one.

2.4.4 Three head beat detector

Outputs of each level of the FPN are passed onto two series of convolutional layers we refer to as *necks*: one convolutional neck for classification, and the

other convolutional neck for regression (see Figure 2.14). Each neck consists of two blocks, each consisting of a 1D convolutional layer with a kernel size of 3 and both input and output channels as 256, a GroupNorm layer (Wu and He 2018), and ReLU layer. This is a common paradigm in object detection; however, instead of using four blocks on each neck, we simplified it to use two instead. The data passes through the classification neck followed by passing it through single convolutional head layer to produce 2-channel classification predictions (see Section 2.4.5), with each channel corresponding to each of the two classes (beat and downbeat). The data is also independently passed through the regression neck in a similar manner, but passes the output of the second block through two 1D convolutional head layers, one with 2-channels (one corresponding to the *left* coordinate and one for the *right* coordinate) and meant to produce regression predictions (see Section 2.4.5), and the other layer with just one channel, corresponding to the leftness score predictions (see Section 2.4.5).

2.4.5 Loss

Given batch size B, total number of anchor point positions per batch item N_k where $k \in \{1, ..., B\}$, classification and regression targets $\mathbf{c}_{k,n}$ and $\mathbf{r}_{k,n}$, and classification and regression predictions $\hat{\mathbf{c}}_{k,n}$ and $\hat{\mathbf{r}}_{k,n}$, the loss used by the model during training can be defined as:

$$L_{\text{total}} = \frac{1}{B} \sum_{k=1}^{B} \left[\frac{1}{N_k} \sum_{n=1}^{N_k} L_{\text{point}}(k, n) + L_{\text{adj}}(\mathbf{c}_{k,1:N_k}, \mathbf{r}_{k,1:N_k}, \hat{\mathbf{r}}_{k,1:N_k}) \right] \tag{2.6}$$

Two components are present in the total loss: the loss calculated per anchor point and the adjacency constraint loss. As explained in further detail in Section

2.4.5, the adjacency constraint loss is unique in that it is calculated given the full set of predictions and targets, while the other losses are calculated per anchor point. The per-point loss component can be calculated as follows:

$$\begin{aligned} L_{\text{point}}(k,n) = & L_{\text{cls}}(\mathbf{c}_{k,n}, \hat{\mathbf{c}}_{k,n}, n) \\ & + \mathbb{1}_{\{\mathbf{c}_{k,n}>0\}} L_{\text{reg}}(\mathbf{r}_{k,n}, \hat{\mathbf{r}}_{k,n}, n) \\ & + \mathbb{1}_{\{\mathbf{c}_{k,n}>0\}} L_{\text{lft}}(\mathbf{r}_{k,n}, \hat{\mathbf{r}}_{k,n}, n) \end{aligned} \tag{2.7}$$

In this equation, L_{cls}, L_{reg}, L_{lft}, and L_{adj} refer to the classification loss, regression loss, and leftness loss respectively. The classification loss is an unchanged version of focal loss proposed with RetinaNet (Lin et al. 2017b) and regression loss is a 1D-simplified version of GIoU loss (Rezatofighi et al. 2019). Leftness loss is simply a left-oriented version of the centerness loss first implemented in FCOS (Tian et al. 2019), and the adjacency constraint loss is a brand new type of loss implemented to penalize training if intervals are not in aligned with one another. $\mathbb{1}_{\{\mathbf{c}_i>0\}}$ is an indicator function that is 1 if the anchor point at position i is positive, and 0 otherwise. Classification, regression, and leftness losses are explained in further detail in sections 2.4.5, 2.4.5, and 2.4.5, respectively.

Classification loss

Focal loss is used in order to prevent the total sum of loss calculated from locations without beat intervals from overwhelming the final loss value, as time points without a beat vastly outnumber locations with a beat interval, an extreme imbalance problem that inspired the implementation of focal loss for object detection (Lin et al. 2017b). When looking to implement the FCOS-based anchor-free approach to our model, discussions were made early on

regarding the feasibility in keeping focal loss. This was because the usage of anchor points at each location would lead to a majority of anchor points resulting in positive due to the fact that beat intervals are expected to be continuously present and adjacent from the very first beat or downbeat to the very last beat or downbeat of the song. However, we eventually agreed to implement a left-based version of the sub-box radius condition added to FCOS later in the 2020 version (Tian et al. 2020) (see Section 2.4.3). This resulted in many more negative anchor points than there would have been without the sub-box implementation, keeping the need for focal loss to handle the class imbalance.

Cross-entropy loss Focal loss is a modified version of the well-known cross-entropy (CE) loss, which is commonly used for the calculation of loss in classification problems. The log function is used in the calculation of CE loss, and produces a high loss value when the probability prediction diverges from its corresponding ground truth and a low loss value otherwise. If there are more than two classification types, it is often referred to as **binary cross-entropy (BCE)** loss. Given the predictions and ground truths $\hat{\mathbf{c}}$ and $\mathbf{c}$ as well as the total number of binary classifiers n, the BCE can be calculated as follows.

$$L_{\mathrm{BCE}}(\mathbf{c}, \hat{\mathbf{c}}, n) = -\frac{1}{n}\sum_{i=1}^{n}\left[\mathbf{c}_i \log(\hat{\mathbf{c}}_i) + (1-\mathbf{c}_i)\log(1-\hat{\mathbf{c}}_i)\right] \tag{2.8}$$

Although the vast majority of problems in object detection have more than two classes, a binary classifier for each class per anchor point is usually used. For example, when dealing with the *COCO* dataset (Lin et al. 2014), there are 80 classes and therefore 80 binary classifiers, and BCE loss is calculated for each. For beat and downbeat tracking, there are two classes and thus two

binary classifiers. Thus, as a binary classifier approaches 1 it can be inferred that the particular class type associated with the binary classifier is detected, and likewise as it approaches 0 it can be inferred that the particular class type is not detected by this anchor point. Therefore, if all binary classifiers associated with a single anchor point are all near 0, this means that no objects are being detected by the anchor point.

Focal loss Although BCE loss is expected to work well in cases where the quantity of items from both classes are identical or similar, an issue occurs when there is class imbalance. Focal loss can essentially be viewed as BCE loss with added weight-adjusting parameters to significantly reduce the influence of loss attributed to easy negatives in problems with extreme class imbalance (e.g. 1:1000) (Lin et al. 2017b). This adjustment resolves the problem of inefficient training (because the search for negative items would therefore be much easier in comparison) and the problem of the sum of each negative item's loss (which would overwhelm the training process). Classification loss L_{cls} is equivalent to focal loss L_{Focal}, which can be defined as follows:

$$L_{\text{Focal}}(\mathbf{c}, \hat{\mathbf{c}}, n) = -\frac{1}{n} \sum_{i=1}^{n} \left[\mathbf{c}_i \alpha (1 - \mathbf{c}_i)^{\gamma} \log(\hat{\mathbf{c}}_i) + (1 - \mathbf{c}_i)(1 - \alpha) \mathbf{c}_i^{\gamma} \log(1 - \hat{\mathbf{c}}_i) \right] \tag{2.9}$$

In the focal loss equation above, two hyperparameters α and γ are introduced. The role of α is to balance the relative weights between positive and negative samples, and the role of γ is to lower the weights of easy examples so that the model can focus on hard examples that the model has a tendency to misclassify during training. In the original implementation, these hyperparam-

eters were set as $\alpha = 0.25$ and $\gamma = 2.0$, which was also used by FCOS (Tian et al. 2019) and us as well.

Because we have two classes in total, our classification loss can be defined by differentiating between class types $\mathbf{c}_{\text{beat}}$, $\mathbf{c}_{\text{downbeat}}$, $\hat{\mathbf{c}}_{\text{beat}}$, and $\hat{\mathbf{c}}_{\text{downbeat}}$, and then using focal loss twice, once per class, as follows:

$$L_{\text{cls}}(\mathbf{c}, \hat{\mathbf{c}}, n) = \frac{1}{2}\left[L_{\text{focal}}(\mathbf{c}_{\text{beat}}, \hat{\mathbf{c}}_{\text{beat}}, n) + L_{\text{focal}}(\mathbf{c}_{\text{downbeat}}, \hat{\mathbf{c}}_{\text{downbeat}}, n)\right] \quad (2.10)$$

Regression loss

Algorithm 1 IoU and GIoU loss algorithm

Input: Bounding boxes $B = (x_1, y_1, x_2, y_2)$, $\hat{B} = (\hat{x_1}, \hat{y_1}, \hat{x_2}, \hat{y_2})$
Output: L_{IoU}, L_{GIoU}

1: Calculate area A of B:
$A = (x_2 - x_1) \times (y_2 - y_1)$
2: Calculate area $\hat{A}$ of $\hat{B}$:
$\hat{A} = (\hat{x}_2 - \hat{x}_1) \times (\hat{y}_2 - \hat{y}_1)$
3: Determine box coordinate variables for calculation of intersection I
$x_1^I = \max(\hat{x_1}, x_1)$, $x_2^I = \min(\hat{x_2}, x_2)$
$y_1^I = \max(\hat{y_1}, y_1)$, $y_2^I = \min(\hat{y_2}, y_2)$
4: Calculate intersection I of bounding boxes B and $\hat{B}$
$$\text{I} = \begin{cases} (x_2^I - x_1^I) \times (y_2^I - y_1^I) & \text{if } x_2^I > x_1^I, y_2^I > y_1^I \\ 0 & \text{otherwise} \end{cases}$$
5: Determine coordinates of smallest enclosing box B_c
$x_1^c = \min(\hat{x_1}, x_1)$, $x_2^c = \max(\hat{x_2}, x_2)$
$y_1^c = \min(\hat{y_1}, y_1)$, $y_2^c = \max(\hat{y_2}, y_2)$
6: Calculate area A_c of smallest enclosing box B_c:
$A_c = (x_2^c - x_1^c) \times (y_2^c - y_1^c)$
7: Calculate union U of bounding boxes B and $\hat{B}$
$U = \hat{A} + A - I$
8: $\text{IoU} = \frac{I}{U}$
9: $\text{GIoU} = \text{IoU} - \frac{A_c - U}{A_c}$
10: $L_{\text{IoU}} = 1 - \text{IoU}$, $L_{\text{GIoU}} = 1 - \text{GIoU}$

Our implementation of the regression loss closely resembles that of FCOS (Tian et al. 2019)(Tian et al. 2020), in which a 1D version of the GIoU loss

(Rezatofighi et al. 2019) is used to learn the normalized left offset $\mathbf{r}_{\text{left}}$ and right offset $\mathbf{r}_{\text{right}}$ from each anchor point corresponding to a position on the base layer of the audio. GIoU is a slight variation of IoU, which is a ratio that ranges from 0 (where there is no overlap at all) to 1 (where the two objects have the same exact size and position). IoU loss allows the model to backpropagate directly on the IoU of two objects, a metric that is widely used in evaluation for object detection. This loss was formulated to address the issues in traditional regression losses for bounding box regression problems, in which an unacceptable box where just two out of four box coordinates are close to the equivalent coordinates of the ground truth box, which can lead to an unjustifiably low loss (Yu et al. 2016). However, the greatest weakness with IoU is that it remains 0 no matter how close or how far apart two boxes are from each other, leading to difficulties in a non-overlapping prediction box from regressing towards the ground truth target box. To resolve this, GIoU loss was introduced, resembling IoU greatly but resulting in growth the further two non-overlapping boxes become. The algorithms for calculating IoU and GIoU losses largely overlap, and is defined in Algorithm 1. As a result, the $L_{\text{reg}}(\mathbf{r}, \hat{\mathbf{r}}, n)$ in Equation 2.6 can be defined as:

$$L_{\text{reg}}(\mathbf{r}, \hat{\mathbf{r}}, n) = 1 - \frac{1}{n} \sum_{i=1}^{n} \text{GIoU}(\mathbf{r}_i, \hat{\mathbf{r}}_i) \tag{2.11}$$

It is important to reiterate that only regression losses for positive anchor points are included, hence the indicator function $\mathbb{1}_{\{\mathbf{c}_i > 0\}}$ in Equation 2.6.

Centerness loss

When a bounding box is predicted, the quality of the predicted bounding box generally correlates with the distance in which it had to regress in order to make

the prediction. Particularly with a point-based model, it makes sense that some of the predictions made would be ample in quantity and from relatively far-off locations. Tian et al. noted this to be a cause for a persistent performance gap with anchor-based models in their first implementation of FCOS (Tian et al. 2019). To improve the quality of the candidate boxes, a centerness branch was added as a single-layer, parallel layer with the classification layer (which was later updated to be parallel with the regression layer instead, resulting in performance improvements (Tian et al. 2020)). Given 2D regression values $\mathbf{r}_{\text{left}}$, $\mathbf{r}_{\text{top}}$, $\mathbf{r}_{\text{right}}$, and $\mathbf{r}_{\text{bottom}}$ for a given location, the 2D centerness target equation provided in the original FCOS paper is defined as:

$$\text{centerness}_{\text{2D}}(\mathbf{r}) = \sqrt{\frac{\min(\mathbf{r}_{\text{left}}, \mathbf{r}_{\text{right}})}{\max(\mathbf{r}_{\text{left}}, \mathbf{r}_{\text{right}})} \times \frac{\min(\mathbf{r}_{\text{top}}, \mathbf{r}_{\text{bottom}})}{\max(\mathbf{r}_{\text{top}}, \mathbf{r}_{\text{bottom}})}} \tag{2.12}$$

Since we are working in 1D, $\mathbf{r}_{\text{top}}$ and $\mathbf{r}_{\text{bottom}}$ is no longer taken into consideration, allowing the equation to be simplified as such:

$$\text{centerness}_{\text{1D}}(\mathbf{r}) = \sqrt{\frac{\min(\mathbf{r}_{\text{left}}, \mathbf{r}_{\text{right}})}{\max(\mathbf{r}_{\text{left}}, \mathbf{r}_{\text{right}})}} \tag{2.13}$$

The result of $\text{centerness}_{\text{1D}}$ ranges between 0 and 1, and is learned using BCE loss. The centerness values are then multiplied with the classification output values, which are defined as M binary classifiers for M classes for N bounding box positions, which by themselves behave as a score for determining of an object is present. As they are multiplied, off-center prediction bounding box weights are decreased.

Left-based centerness (leftness) For our model, we created a left-based version of the centerness branch introduced in FCOS (Tian et al. 2019), referred

to here as leftness. As the name implies, the score is highest on the left side of the beat interval and weakest on the right. We also implemented the radius mentioned in the second FCOS paper (Tian et al. 2020) so that only positions existing within the radius at the left side of the beat interval are selected as positive anchor points. The reason for causing the model to focus on the leftmost part of the beat interval instead of the center is that the left is where the point in which the actual beat is present. This is counterintuitive when looking strictly from the perspective of object detection, and more intuitive when understanding that the the left offset l can instead be seen as the offset from the current point to the actual beat, and the right offset r as the estimated time in which the next beat appears. It thus makes sense to focus training on the left part. The leftness equation is as follows:

$$\text{leftness}_{\text{1D}}(\mathbf{r}) = \sqrt{\frac{\mathbf{r}_{\text{right}}}{\mathbf{r}_{\text{left}} + \mathbf{r}_{\text{right}}}} \tag{2.14}$$

As FCOS used binary cross-entropy loss for their centerness, we also use the same type of loss for leftness, and so this can be defined as follows:

$$L_{\text{lft}}(\mathbf{r}, \hat{\mathbf{r}}, n) = L_{\text{BCE}}(\text{leftness}_{\text{1D}}(\mathbf{r}), \text{leftness}_{\text{1D}}(\hat{\mathbf{r}}), n) \tag{2.15}$$

L_{lft} in Equation 2.7 can be substituted with L_{ctr}, which would likewise be defined as:

$$L_{\text{ctr}}(\mathbf{r}, \hat{\mathbf{r}}, n) = L_{\text{BCE}}(\text{centerness}_{\text{1D}}(\mathbf{r}), \text{centerness}_{\text{1D}}(\hat{\mathbf{r}}), n) \tag{2.16}$$

Algorithm 2 Adjacency constraint loss

procedure CONSTRAIN($B1$, $B2$, a, b, d)
    $L \leftarrow 0$
    for $b_i \in B1$ **do**
        $t_{a,i} \leftarrow x_a$ of b_i
        for $b_j \in B2$ **do**
            $t_{b,j} \leftarrow x_b$ of b_j
            if $b_i = b_j$ **then**
                $\hat{t}_{a,i} \leftarrow \hat{x}_a$ of $\hat{b}_i$
                $\hat{t}_{b,j} \leftarrow \hat{x}_b$ of $\hat{b}_j$
                $L \leftarrow L + ((\hat{t}_{a,i} - \hat{t}_{b,j})/d)^2$ ▷ MSE between $\hat{t}_{a,i}/d$ and $\hat{t}_{b,j}/d$
            end if
        end for
    end for
    return L
end procedure
$d_1 \leftarrow \max(T_{downbeat} \cup T_{beat}) - \min(T_{downbeat} \cup T_{beat})$
$d_2 \leftarrow \max(T_{downbeat}) - \min(T_{downbeat})$
$d_3 \leftarrow \max(T_{beat}) - \min(T_{beat})$
$L_{total} \leftarrow L_{total} + \text{constrain}(B_{downbeat}, B_{beat}, 1, 1, d_1)$
$L_{total} \leftarrow L_{total} + \text{constrain}(B_{downbeat}, B_{downbeat}, 2, 1, d_2)$
$L_{total} \leftarrow L_{total} + \text{constrain}(B_{beat}, B_{beat}, 2, 1, d_3)$

Adjacency constraint loss

One problem we experienced in the application of object detection model techniques with beat detection was that the existing loss methods do not guarantee that beats and downbeats will be aligned with one another. This is not normally a problem in traditional object detection, in which the position of all detected objects are generally independent of one another as there is neither requirement nor expectancy of adjacency. Furthermore, this issue was more present in the downbeats, each of which are represented by two different interval objects, one as a short beat interval spanning from the itself to the next beat and the other as a long downbeat interval spanning from itself to the next downbeat. Since, even though there are technically two separate objects,

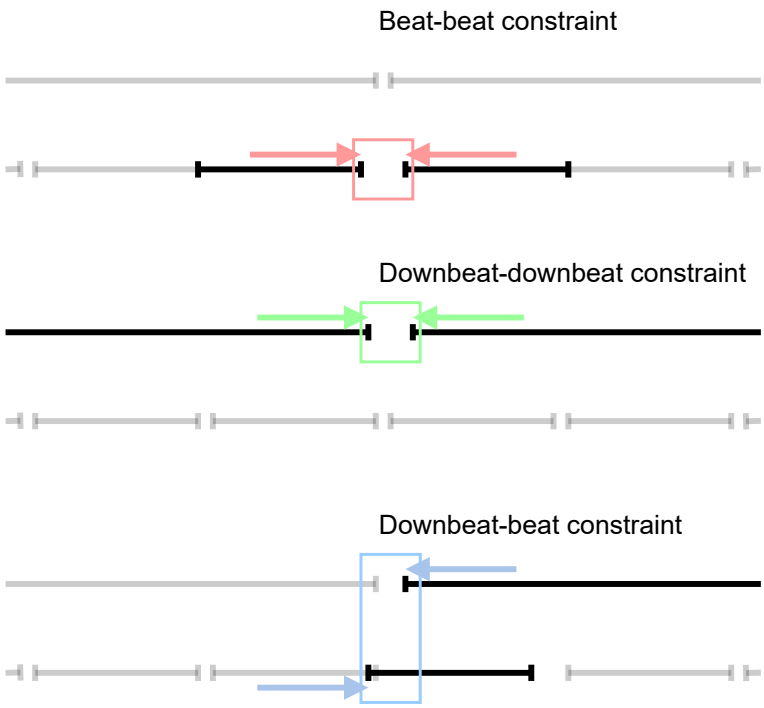

Figure 2.15 Illustration of the adjacency constraint loss in action. Three combinations of intervals are *constrained*: beats-beats (red), downbeats-downbeats (green), and beats-downbeats (blue).

ultimately this is just one downbeat represented twice, these should share the same start times. In other words, every downbeat interval's starting time should share the same starting point as its corresponding beat, but often times there was little to none of such correlation whatsoever.

This is a brand new problem that does not exist in traditional object detection models. To resolve this, we implemented the brand new **adjacency constraint loss** (see Algorithm 2). This is the fourth loss that is added to the total loss of the model after the classification, regression, and centerness (leftness) losses that exist in present object detection models. This loss is also unique in that the calculation of it depends on the predictions of multiple

anchor points, unlike the other three losses that are calculated per anchor point.

The goal of this loss is threefold: first, to minimize the distance between a beat interval's ending point and the next beat interval's starting point; second, to minimize the distance between a downbeat interval's ending point and the next downbeat interval's starting point; and third, to minimize the distance between the starting points of a downbeat interval and its corresponding regular beat interval. We use mean squared error (MSE) to calculate the errors, normalizing the beat interval time points by dividing the difference of time points by the total effective length of the audio, that is, the total distance between the last beat or downbeat and the first beat or downbeat in the audio. MSE loss, also known as L2 loss, is a type of regression loss that is calculated by squaring the difference of two points (see Equation 2.17). As this grows exponentially with increase of distance, it is more punishing of outliers and also smoothly converges to 0.

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \tag{2.17}$$

2.4.6 Non-maximum suppression

Figure 2.16 Illustration of one-dimensional non-maximum suppression taking place on a set of beat interval predictions.

After object candidate proposals are produced, postprocessing must be done to ensure that a single object is not represented by more than one predicted

bounding box. The reason is that one object can and will have multiple anchor points assigned to it which will lead to multiple predictions of the same item, as there is no built-in logic in its pure implementation that prevents itself from doing so. Since it is expected to have multiple overlapping boxes pointing to a single object on an image, the non-maximum suppression (NMS) algorithm is commonly used to filter out overlapping and redundant boxes with the objective of leaving behind only one box pointing to each object with the highest confidence score (see Figure 2.16).

This algorithm (see Algorithm 3) takes a list of boxes detected by the model as input along with a corresponding confidence score for each box and the order in which the boxes are processed is based on descending order of confidence score. For each detected box, all other boxes that are also located at a similar region are suppressed (i.e. removed) from the detection box list. The region is generally defined by checking the IoU of the two boxes, and determined to be within a similar region if this value rises over a certain threshold value. The lower the IoU threshold value is, the more aggressive suppression becomes.

It is important to mention that the usage of NMS by object detection models, which includes our model as well, it is difficult to claim that we follow an end-to-end approach. However, the appeal is that NMS simply takes in the data and filters *redundant* predictions, instead of reconstructing a brand new set of data after verifying it with a set of relatively complex checks that are based on more domain knowledge in respect to music theory.

Soft-NMS Although non-maximum suppression continues to be widely used, a major weakness to the object detection paradigm is its lack of ability in differentiating when overlapping boxes point to a single object or to multiple

Algorithm 3 Non-maximum suppression algorithm

Input: Bounding boxes B sorted in descending order by C
Input: Classification scores C where $c_i \in C$ corresponds to $b_i \in B$
Output: NMS-filtered bounding boxes $B_{filtered}$

$B_{filtered} \leftarrow \emptyset$
for $b_i \in B$ **do**
    $discard \leftarrow$ False
    for $b_j \in B$ **do**
        if IoU$(b_i, b_j) > \lambda_{nms}$ **then** ▷ λ_{nms} is typically 0.5
            if $c_j > c_i$ **then**
                $discard \leftarrow$ True
            end if
        end if
    end for
    if $discard =$ False **then**
        $B_{filtered} \leftarrow B_{filtered} \cup b_i$
    end if
end for

objects of the same class. A very common example of this is when a photo of a densely packed crowd of people is passed into an object detection model. If each person overlaps greatly with other people, the traditional NMS algorithm would be expected to perform poorly. This led to the introduction of Soft-NMS, a slight modification of the original NMS algorithm that simply lowers the weights of lower scored overlapping boxes instead of removing them entirely (Bodla et al. 2017). This slight variation of NMS has shown to be effective for instances in which the number of false positive predictions are higher. If the original NMS algorithm is reframed as a rescoring algorithm as opposed to a pruning algorithm as demonstrated in Algorithm 3 such that each box is given a new score, then this rescoring function can be expressed as follows:

$$c_j = \begin{cases} c_j & \text{if IoU}(b_i, b_j) < \lambda_{nms} \text{ or } c_i < c_j \\ c_{j,new} & \text{otherwise} \end{cases} \tag{2.18}$$

For the original NMS, $c_{j,new}$ would simply be 0, taking into consideration that pruning the box is equivalent to setting its score to 0 and removing all boxes with scores equal to (or very close to) 0. However as aforementioned, Soft-NMS simply lowers these scores instead of removing them completely. The original version of $c_{j,new}$ in Soft-NMS was $c_{j,new} = c_j(1 - \text{IoU}(b_i, b_j))$ but this version decays the scores proportionately to the amount of overlap but is not continuous and has a sudden penalty once the NMS threshold is reached, was eventually revised with the added usage of a Gaussian penalty function defined as:

$$c_{j,new} = c_j e^{\frac{\text{IoU}(b_i, b_j)^2}{\sigma}} \tag{2.19}$$

2.4.7 Datasets

Datasets used in WaveBeat and BeatFCOS				
Dataset	# of files	Total duration	Beats	Downbeats
Ballroom	685	5h 57m	✓	✓
Hainsworth	222	3h 19m	✓	✓
RWC Popular	100	6h 46m	✓	✓
Beatles	179	8h 00m	✓	✓
GTZAN	993	8h 17m	✓	✓
SMC	217	2h 25m	✓	

Table 2.1 Datasets used for both WaveBeat and BeatFCOS. The *GTZAN* and *SMC* datasets were held out entirely and used solely for testing.

We used the same datasets as WaveBeat (Steinmetz and Reiss 2021), which includes *Ballroom* (Gouyon et al. 2006)(Krebs et al. 2013), *Hainsworth* (Hainsworth and Macleod 2004), *Beatles* (Davies et al. 2009)(Harte 2010), and *RWC Popular* (Goto et al. 2002) for training datasets, as well as *GTZAN* (Tzanetakis and Cook 2002)(Marchand and Peeters 2015) and *SMC* (Holzapfel et al. 2012) as test datasets. Beat tracking data retrieval is very difficult; unlike popular datasets like *COCO* (Lin et al. 2014), there is no way to download the same exact data from one place online. Thus, it is very much possible that the dataset being used by each author is not identical, and so we decided to train WaveBeat on our own using the datasets in our possession in order to make a more fair and apt comparison. The paper explaining the TCN model mentioned that duplicate audio files in the *Ballroom* dataset discovered by

Bob Sturm[2] were removed, and so we did the same. This was also the case for WaveBeat (Steinmetz and Reiss 2021); even though it was not mentioned explicitly, the total duration of the Ballroom dataset that was mentioned is consistent with our calculated total duration, indicating that they also removed the duplicate files. The *GTZAN* dataset normally consists of 1000 items, but the items without downbeat labels were excluded, resulting in a total of 993 items that we used to evaluate the model. All of the beat and downbeat interval lengths were calculated in advance using the k-means algorithm to determine regression limits for the anchor points (see Section 2.4.3).

Difficulty in *Beatles* dataset collection

Among the collected datasets, the *Beatles* dataset was particularly difficult to collect. Although the beat annotation data is easy to find online, they correspond to the original *Beatles* albums from the 1960s and 1970s (Harte 2010). Since the release of these original albums, subsequent versions of the same music have been and continue to be released with refinements and improvements to the audio. These changes seem to also include adjustments to the audio lengths themselves which results in shifting of the actual song, causing beats to be misaligned entirely. This problem was discovered when the *Beatles* music data we had collected earlier on was causing problems to the training process due to the misaligned beats. Eventually we were able to collect the original music audio that properly corresponds with the beat annotations, which was verified by simply running the entire dataset through validation of an existing beat tracker. Thus, we see it to be a concern that some researchers would test

[2]https://highnoongmt.wordpress.com/2014/01/23/ballroom_dataset/

their work using the wrong music under the false but easy-to-make assumption that all *Beatles* music is the same.

2.5 Experiments

2.5.1 Evaluation

Every time a neural network finishes an epoch of training (i.e. after the neural network learns from every item in the training dataset once), the performance of the model is evaluated using the evaluation dataset, which consists of data that was unseen during training. This is to ensure that the model does not memorize irrelevant features specific to the training data but is instead able to produce expected results even when looking at unseen data. To measure the performance of a model, evaluation metrics are used during evaluation mode. Comparison of loss values and concluding that the model improves by simply looking at the loss value can be done, but the loss value is meant to calculate how the model's weights should backpropagate, and is not the clearest indicator of how well the model can do in a certain task.

Definitions

Confusion matrix A confusion matrix is a measurement commonly used in classification-based problems. In the case of machine learning, it would summarize the performance of a model where, given some data, the predictions that are made are compared with the corresponding ground-truth values and are categorized into four categories: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). These four can be simply defined as follows, along with examples that apply to beat tracking:

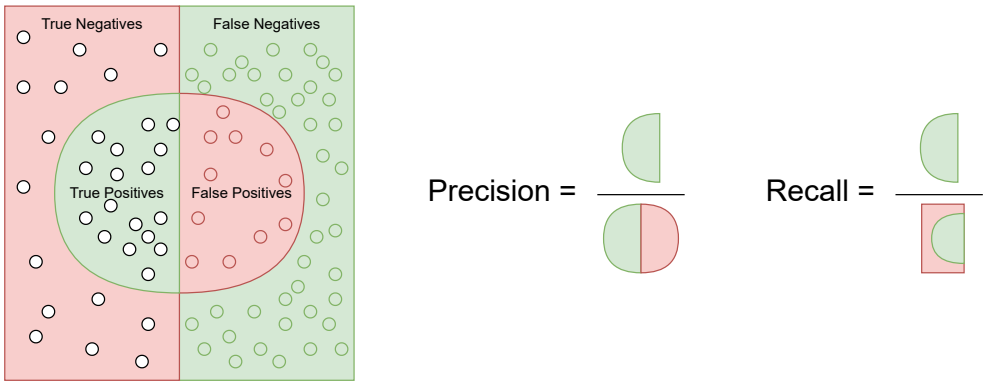

Figure 2.17 A visual illustration of precision and recall given data points separated into four categories of the confusion matrix.

- True Positive (TP) - Refers to correct positive classifications, e.g. a moment in time has a beat that was correctly predicted
- False Positive (FP) - Refers to incorrect positive classifications, e.g. a moment in time has no beat but a prediction was made that indicates otherwise
- True Negative (TN) - Refers to correct negative classifications, e.g. a moment in time has no beat and so no beat prediction was made
- False Negative (FN) - Refers to incorrect negative classifications, e.g. a moment in time has a beat but was not predicted

After the determination of the proper category of each prediction, several metrics can be calculated and used to evaluate the performance of models which has several purposes. One purpose is that the improvement of the neural network as training progresses can be tracked which can determine the tweaking of hyperparameters as time progresses. Another purpose is that different models that perform similar tasks can be benchmarked using a common evaluation metric which would allow for simple comparisons to be made.

Precision Precision (see the left equation in Figure 2.17) refers to the fraction of the number of true positives and the number of all positive predictions (the sum of true and false positives) made. In other words, this fraction describes how good the predictor is in making the correct positive prediction. For example, the prediction value in beat tracking would be the number of correct beats divided by the number of all predicted beats. However, this value does not take false negatives into account at all. If the predictor makes only one prediction in total but it happens to be correct, this would still result in a perfect prediction score. It can be defined in an equation as follows:

$$p = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2.20}$$

Recall Recall (see the right equation in Figure 2.17) refers to the fraction of the number of true positives and the total number of true values (the sum of true positives and false negatives). In other words, this fraction describes how good the predictor is in detecting all true values. For example, the recall value in beat tracking would be the number of correct beats divided by the number of all actual beats. However, this value does not take false positives into account at all. If the predictor maximizes its number of predictions, this would still result in a perfect recall score. It can be defined in an equation as follows:

$$r = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.21}$$

Evaluation metrics

F-measure The F-measure score is a generic measure that indicates the accuracy of a test. It is calculated using three parameters: the number of true positives, the number of false negatives, and the number of false positives, in which beats or downbeats are considered to be correctly detected if the time position falls within a tiny window of time, which is usually ±70 milliseconds in beat tracking (Davies et al. 2009). It can be calculated as follows:

$$F = \frac{2pr}{p+r} = \frac{2 * \mathrm{TP}}{2 * \mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{2.22}$$

Due to the familiarity of F-measure, it is probably the most commonly used measure for beat tracking, but there are shortcomings to this approach. For example, if one predicted beat is 70 milliseconds too early and the preceding beat is 70 milliseconds too late, this error will be noticeable to humans if these predictions cause some additional behavior (e.g. if tick noises are overlaid on top of a song where beats are present) yet the F-measure score will not consider this to be a problem. Although the F-measure score is effective in getting a general idea regarding the performance of a model, there are still several shortcomings that provide the necessity to refer to multiple types of metrics when comparing models. This also includes the fact that beats are not measured in sequences but instead *time-independent*, or measured independently from one another. Another consideration to make is that the F-measure can result in a score of 0, but this does not tell us if the predicted beats have a totally random relationship with the ground truth time points, or if there is some sort of meaningful pattern to the predictions (for example, if the beats were predicted on the *off-beat*) (Matthew E. P. Davies 2021).

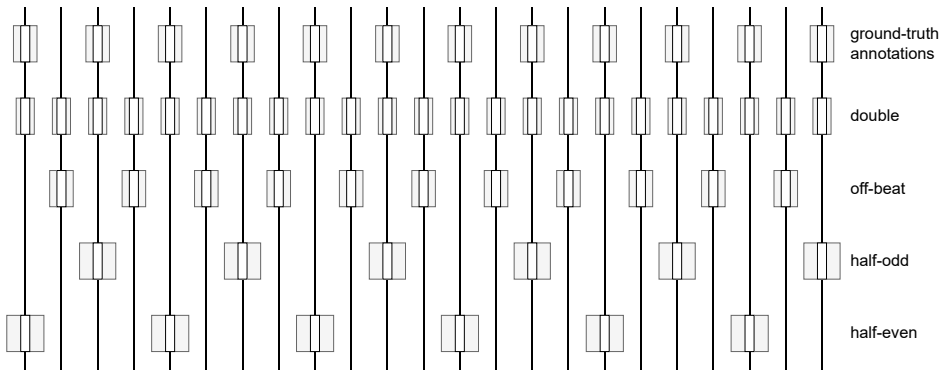

Figure 2.18 Some evaluation metrics check if predicted beats match more than just the set of ground-truth annotations, using multiple metrical variations and using the best score. The white rectangles represent beats, and the gray rectangles represent the tolerance windows that determine whether or not a beat is correctly predicted.

Continuity-based evaluation Some beat evaluation metrics take continuity of correct beats into consideration, requiring not only beat i but also beat $i-1$ to be accurate when calculating the score (Davies et al. 2009). Another added feature is the evaluation not only of the ground-truth beat annotations but also a number of variants (see Figure 2.18). The reason behind this is that there are many scenarios in which the beat tracking model detects the off-beats or some other beat pattern, which is much different from making random predictions that have no clear relationship to the actual beats in the music. As it can be seen in Figure 2.18, tolerance windows are not fixed at 70 milliseconds but instead are calculated in a beat-relative manner of $\pm 17.5\%$ (Davies et al. 2009). Finally, the score can be calculated by finding the ratio of the longest correct segment to the length of the excerpt, which is the *CMLc* or *AMLc*, or the ratio of the total of correct segments to the length of the excerpt, which is the *CMLt* or *AMLt*. CML simply refers to the *correct* metrical level which is the original beat ground-truth evaluation metric, and AML refers to the max of all the *allowed* metrics shown in Figure 2.18.

Mean Average Precision Mean Average Precision (mAP) is a metric commonly used for the evaluation of object detection models. It can be produced by calculating the mean of the average precision per object type. Each average precision can be found by simply plotting the precision and recall on a two dimensional graph, drawing a curve that passes through the plot, and calculating the area under the curve. A large area indicates great performance of the model in making predictions. In mAP, True Positives are determined simply by checking if the IoU between the prediction and ground truth bounding boxes exceeds a certain threshold. This is usually set to 0.5, but more recently mAP been measured and averaged multiple times at different IoU thresholds at [0.5 : 0.05 : 0.95] (Huang et al. 2017).

Because calculation of mAP relies the ability to calculate IoU for the given predictions, it is not possible to measure traditional beat tracking models using it without first retrieving the final detected and ground-truth timepoints and converting them to intervals before applying them with 1D IoU. However, because BeatFCOS is designed to predict intervals, mAP can be used to measure its performance.

2.5.2 Training

All results reported in Section 2.5.3 have all been trained using the Adam optimizer with a learning rate of $1e^{-3}$ and weight decay of $1e^{-4}$, decreasing by a factor of 10 if the joint F-measure score (the average of the beat and downbeat F-measure scores) sees no improvement for three epochs. Although WaveBeat used a patience of 10 epochs, we saw greater results with a lower number, most likely because we were primarily working with pretrained backbones that converge much faster. We set the batch size to 16 and performed all training

on Google Colab instances, each with a single NVIDIA A100 40GiB GPU. We also followed the approach in WaveBeat (Steinmetz and Reiss 2021) by loading audio at 22.05 kHz, changing the length of the audio to always fit to $2^{21} = 2097152$ samples ($\approx$ 1.6 minutes), padding or cutting when necessary, and made each dataset represent 1000 music excerpts per epoch for a total of 100 epochs, in order to prevent one dataset from dominating another in representation during the training process. However, unlike WaveBeat, we did not clip our gradients.

2.5.3 Results

8-fold validation test for WaveBeat

The original paper for WaveBeat (Steinmetz and Reiss 2021) displays comparisons when using peak picking versus a DBN for post-processing, as well as a Spectral TCN referring to the model and scores reported in (Böck and Davies 2020) for benchmarking purposes. However, the scores pertaining to WaveBeat were noticeably calculated with a simple 80/10/10 split, whereas the Spectral TCN scores were calculated using 8-fold validation. This, along with the fact that the distribution of beat annotation data is not centralized, opening the possibility that the dataset used to train our model differs slightly from theirs, thereby providing valid reason to retrain the WaveBeat model. 8-fold validation was performed using the folds defined in the GitHub repository provided in (Böck and Davies 2020)[3]. For the *RWC Popular* dataset which was not used by them, the folds were simply defined by calculating the modulus of the track number (which ranges from 1 to 100) by 8, the number of folds, and is provided

[3]https://github.com/superbock/ISMIR2020/tree/master/splits

on GitHub[4]. For the WaveBeat backbone, the original hyperparameters as defined in the paper (Steinmetz and Reiss 2021) were kept as-is, and was trained nine times in total: trained eight times for each of the folds, and trained once for the single 80/10/10 split.

[4]https://github.com/zaiisao/beatfcos-reference-files

WaveBeat with 8-fold validation							
		Beat			Downbeat		
Dataset	Type	F1	CMLt	AMLt	F1	CMLt	AMLt
Ballroom	WaveBeat, Peak (8-fold)	0.896	0.792	0.820	0.687	0.339	0.606
	WaveBeat, Peak (80/10/10)	0.925	0.836	0.845	0.750	0.388	0.677
	WaveBeat, Peak (80/10/10 (Steinmetz and Reiss 2021))	0.961	0.929	0.929	0.904	0.762	0.803
	WaveBeat, DBN (8-fold)	0.864	0.711	0.900	0.748	0.563	0.853
	WaveBeat, DBN (80/10/10)	0.910	0.798	0.933	0.800	0.592	0.892
	WaveBeat, DBN (80/10/10 (Steinmetz and Reiss 2021))	0.925	0.829	0.937	0.953	0.916	0.941
Hainsworth	WaveBeat, Peak (8-fold)	0.755	0.609	0.662	0.466	0.182	0.388
	WaveBeat, Peak (80/10/10)	0.902	0.832	0.843	0.711	0.314	0.523
	WaveBeat, Peak (80/10/10 (Steinmetz and Reiss 2021))	0.965	0.937	0.937	0.912	0.748	0.843
	WaveBeat, DBN (8-fold)	0.778	0.712	0.829	0.509	0.287	0.643
	WaveBeat, DBN (80/10/10)	0.900	0.882	0.916	0.782	0.544	0.872
	WaveBeat, DBN (80/10/10 (Steinmetz and Reiss 2021))	0.973	0.976	0.976	0.954	0.886	0.970
Beatles	WaveBeat, Peak (8-fold)	0.886	0.735	0.815	0.685	0.330	0.544
	WaveBeat, Peak (80/10/10)	0.896	0.723	0.870	0.758	0.455	0.704
	WaveBeat, Peak (80/10/10 (Steinmetz and Reiss 2021))	0.887	0.733	0.790	0.689	0.327	0.585
	WaveBeat, DBN (8-fold)	0.893	0.786	0.901	0.758	0.473	0.831
	WaveBeat, DBN (80/10/10)	0.848	0.720	0.934	0.803	0.531	0.904
	WaveBeat, DBN (80/10/10 (Steinmetz and Reiss 2021))	0.929	0.894	0.894	0.732	0.509	0.724
RWC Popular	WaveBeat, Peak (8-fold)	0.836	0.681	0.755	0.646	0.336	0.483
	WaveBeat, Peak (80/10/10)	0.978	0.931	0.931	0.913	0.763	0.815
	WaveBeat, Peak (80/10/10 (Steinmetz and Reiss 2021))	–	–	–	–	–	–
	WaveBeat, DBN (8-fold)	0.864	0.771	0.905	0.692	0.442	0.793
	WaveBeat, DBN (80/10/10)	0.976	0.943	0.943	0.905	0.940	0.935
	WaveBeat, DBN (80/10/10 (Steinmetz and Reiss 2021))	–	–	–	–	–	–
GTZAN	WaveBeat, Peak (8-fold)	0.809	0.644	0.723	0.520	0.175	0.458
	WaveBeat, Peak (80/10/10)	0.810	0.647	0.730	0.523	0.181	0.464
	WaveBeat, Peak (80/10/10 (Steinmetz and Reiss 2021))	0.825	0.682	0.767	0.563	0.279	0.515
	WaveBeat, DBN (8-fold)	0.831	0.716	0.847	0.567	0.320	0.730
	WaveBeat, DBN (80/10/10)	0.828	0.711	0.868	0.570	0.315	0.743
	WaveBeat, DBN (80/10/10 (Steinmetz and Reiss 2021))	0.828	0.719	0.860	0.598	0.503	0.764
SMC	WaveBeat, Peak (8-fold)	0.413	0.167	0.250	–	–	–
	WaveBeat, Peak (80/10/10)	0.409	0.163	0.245	–	–	–
	WaveBeat, Peak (80/10/10 (Steinmetz and Reiss 2021))	0.403	0.163	0.255	–	–	–
	WaveBeat, DBN (8-fold)	0.431	0.288	0.431	–	–	–
	WaveBeat, DBN (80/10/10)	0.435	0.303	0.429	–	–	–
	WaveBeat, DBN (80/10/10 (Steinmetz and Reiss 2021))	0.418	0.280	0.419	–	–	–

Table 2.2 Results from WaveBeat in which peak-picking and DBN are compared. Scores were reported using both checkpoints trained with 8-fold validation and 80/10/10 split. Also included are the 80/10/10 split scores from the original paper (Steinmetz and Reiss 2021).

Table 2.2 compares the three sets of WaveBeat DBN and peak-picking test results: one using 8-fold verification, one retrained using our datasets and 80/10/10 splits[5], and one with the scores reported in the original paper (Steinmetz and Reiss 2021), in which several observations were made. The biggest observation is that it is not fair to treat 8-fold verification scores the same as 80/10/10 split as they can each vary greatly; especially with smaller datasets with more diversity, scores calculated from single 80/10/10 split will overestimate the efficacy of the model. This also resolves an unanswered question regarding the unusually high beat and downbeat scores in the *Hainsworth* dataset, which surpasses even the SOTA scores provided by the Transformer-based model by Hung et al., (Hung et al. 2022) an unusual observation. It is also important to mention that a major bug was discovered in the original WaveBeat code during this experiment, causing many files in the dataset to be included in both the training and verification subsets, causing an artificial inflation in the verification scores and leading to incorrect hyperparameter fine-tuning during the training process. The reevaluated scores in the table on a checkpoint file that was trained after this bug was fixed. The lower test scores after fixing indicate spectrograms are here to stay for the time being. However, using raw audio to train beat tracking models can potentially become a reality once the issue surrounding the lack of labeled beat data is resolved,

[5]The exact 80/10/10 splits can be found in this GitHub repository: https://github.com/zaiisao/beatfcos-reference-files

and WaveBeat demonstrates that usage of data augmentation can improve results quite significantly in the case raw audio is used to train the model. The existence of this bug as well as the 8-fold results have been confirmed and approved by Steinmetz.

Added features to BeatFCOS

As covered in Section 2.4, there were a number of design decisions that were made and testing was performed for each significant choice to determine the most optimal implementation. Table 2.3 compares results using centerness vs. leftness, as well as the adjacency constraint. We have established with the retested results of WaveBeat using 8-fold validation (see 2.5.3) that it is not reasonable to compare 8-fold scores with 80/10/10 split scores, and that using 8-fold provides results that most accurately represent the performance of the model, especially when all the results follow the same splits, a condition that has been met in the collection of these scores. Usage of leftness shows clear benefits and usage with adjacency constraints does lead to best results in a plurality of cases, but surprisingly it is sometimes outperformed by the version without adjacency constraints. This appears to indicate that it causes the performance to drop slightly in some cases, and always falls short when measuring on the AMLt scores. Nevertheless, due to its slight edge, we decided against removing it for the remainder of tests.

BeatFCOS with Leftness and Adjacency Constraint Loss								
			Beat			Downbeat		
Dataset	LEFT	ADJ	F1	CMLt	AMLt	F1	CMLt	AMLt
Ballroom			**0.926**	**0.862**	**0.885**	0.622	0.450	0.587
	✓		0.913	0.797	0.818	0.783	0.604	**0.691**
		✓	0.911	0.835	0.860	0.640	0.499	0.582
	✓	✓	0.924	0.840	0.860	**0.795**	**0.641**	0.689
Hainsworth			0.739	0.641	0.716	0.513	0.389	0.490
	✓		0.833	0.709	0.760	**0.684**	**0.540**	**0.629**
		✓	0.750	0.621	0.704	0.536	0.421	0.528
	✓	✓	**0.834**	**0.726**	**0.777**	0.661	0.522	0.601
Beatles			0.974	0.951	0.951	0.815	0.667	0.690
	✓		0.977	0.951	0.951	**0.908**	**0.798**	**0.837**
		✓	0.970	0.943	0.943	0.808	0.645	0.691
	✓	✓	**0.980**	**0.963**	**0.963**	0.889	0.787	0.815
RWC Popular			0.960	0.919	0.919	0.915	0.855	0.866
	✓		0.971	0.941	0.941	**0.947**	0.871	0.871
		✓	0.957	0.915	0.915	0.901	0.852	0.852
	✓	✓	**0.984**	**0.973**	**0.973**	0.916	**0.890**	**0.890**
GTZAN			0.787	0.641	0.733	0.445	0.270	0.462
	✓		0.782	0.635	0.733	**0.514**	0.336	**0.492**
		✓	**0.790**	**0.649**	0.738	0.448	0.292	0.466
	✓	✓	0.787	0.647	**0.744**	0.512	**0.342**	0.485
SMC			0.402	0.245	0.311	–	–	–
	✓		**0.411**	**0.252**	**0.324**	–	–	–
		✓	0.403	0.241	0.315	–	–	–
	✓	✓	0.399	0.241	0.302	–	–	–

Table 2.3 Comparison of BeatFCOS versions. Here, centerness and leftness is compared, as well as the adjacency constraint loss. All scores here were evaluated using an 80/10/10 split, with each checkpoint trained, validated, and tested using the same exact split. Results without LEFT checkmarked were trained using centerness as opposed to leftness. ADJ refers to adjacency constraint loss.

NMS vs. Soft-NMS

As seen in Table 2.4, usage of Soft-NMS improves scores significantly. The reason seems to be quite simple, in that Soft-NMS was designed to work well in crowded images of the same class, and when portraying beats and downbeats as adjacent intervals throughout the input audio data, this problem can be seen as analogous with a crowded image with many objects of the same class grouped together. Because of this, we used Soft-NMS by default for all other evaluation.

BeatFCOS with NMS and Soft-NMS							
		Beat			Downbeat		
Dataset	Type	F1	CMLt	AMLt	F1	CMLt	AMLt
Ballroom	NMS	0.901	0.811	0.816	0.737	0.484	0.656
	Soft-NMS	**0.924**	**0.840**	**0.860**	**0.795**	**0.641**	**0.689**
Hainsworth	NMS	**0.836**	**0.734**	0.774	0.640	0.421	0.554
	Soft-NMS	0.834	0.726	**0.777**	**0.661**	**0.522**	**0.601**
Beatles	NMS	0.949	0.897	0.897	0.800	0.621	0.694
	Soft-NMS	**0.980**	**0.963**	**0.963**	**0.889**	**0.787**	**0.815**
RWC Popular	NMS	0.953	0.922	0.922	0.897	0.806	0.824
	Soft-NMS	**0.984**	**0.973**	**0.973**	**0.945**	**0.890**	**0.890**
GTZAN	NMS	0.781	0.616	0.678	0.488	0.227	0.445
	Soft-NMS	**0.787**	**0.647**	**0.744**	**0.512**	**0.342**	**0.485**
SMC	NMS	**0.409**	0.203	0.261	–	–	–
	Soft-NMS	0.399	**0.241**	**0.302**	–	–	–

Table 2.4 Comparison of scores when using NMS versus Soft-NMS for post-processing of the beat and downbeat intervals. Scores were reported using a checkpoint with leftness and adjacency constraint enabled, trained with 80/10/10 split and Soft-NMS for validation.

Frozen backbone

Experiments were also conducted to compare performance with and without freezing the weights of the WaveBeat backbone when training BeatFCOS. Although freezing the weights would cause less potential for optimization to work more efficiently with the BeatFCOS architecture, experiment is particularly meaningful due to the role and objective of BeatFCOS as an alternative to the dominant DBN approach for post-processing, which is not simultaneously

trained with the deep neural networks it links with. As seen in Table 2.5, performance of non-frozen backbone is superior for all scores.

		Beat			Downbeat		
Dataset	Frozen	F1	CMLt	AMLt	F1	CMLt	AMLt
Ballroom	Entire backbone	0.878	0.771	0.815	0.703	0.513	0.603
	BatchNorm only	**0.927**	**0.873**	**0.898**	**0.807**	**0.697**	**0.756**
Hainsworth	Entire backbone	0.740	0.628	0.675	0.451	0.317	0.410
	BatchNorm only	**0.761**	**0.678**	**0.735**	**0.529**	**0.416**	**0.500**
Beatles	Entire backbone	0.888	0.762	0.830	0.709	0.474	0.566
	BatchNorm only	**0.903**	**0.797**	**0.866**	**0.762**	**0.579**	**0.659**
RWC Popular	Entire backbone	0.837	0.710	0.771	0.664	0.487	0.542
	BatchNorm only	**0.862**	**0.763**	**0.849**	**0.779**	**0.691**	**0.731**
GTZAN	Entire backbone	0.782	0.628	0.712	0.495	0.302	0.455
	BatchNorm only	**0.808**	**0.682**	**0.773**	**0.546**	**0.378**	**0.543**
SMC	Entire backbone	0.392	0.213	0.279	–	–	–
	BatchNorm only	**0.400**	**0.244**	**0.315**	–	–	–

Table 2.5 Comparison of scores when freezing the backbone and just freezing the batch normalization layers. All scores were reported using checkpoints trained with 8-fold validation.

Comparison with other models

In Table 2.6, we compare three versions of WaveBeat: the peak picking and DBN results from the original WaveBeat retrained and tested by us using 8-fold validation (see Section Table 2.2), as well as a version of WaveBeat used as a backbone for BeatFCOS and the greatest scores among the three versions of WaveBeat are bolded. Additionally, the Spectral TCN model (Böck and Davies 2020) benchmarked in the original WaveBeat paper (Steinmetz and Reiss 2021), as well as the new SOTA model by Hung et al. (Hung et al. 2022)

were also included in the table for reference purposes, but were excluded from the score bolding to allow for more ease in seeing which version of WaveBeat performs the best.

When matched up with the DBN version of WaveBeat, the BeatFCOS version appears to lag behind slightly in beat prediction but have a noticeable edge in downbeat prediction. However, an interesting observation is that the downbeat AMLt score for the DBN version always surpasses its BeatFCOS counterpart. After taking into consideration how AMLt is calculated (see Section 2.5.1) as well as how DBNs work (see Section 2.3.2), it can be inferred from the data that the DBN is producing a realistic yet misaligned set of beats from data that is lacking in quality. The ability of DBNs to produce very realistic sets of beat positions coupled with the leniency of AMLt with misalignments, it ultimately makes sense that DBNs would perform well when evaluating with AMLt. Another observation with BeatFCOS is that it seems to perform quite well with the four train datasets but underperform quite significantly with the held out test datasets. Although the CMLt and AMLt scores for BeatFCOS surpass those of the WaveBeat version that uses pick picking, the F-measure score is quite low in comparison.

		Beat			Downbeat		
Dataset	Model	F1	CMLt	AMLt	F1	CMLt	AMLt
Ballroom	WaveBeat (Peak)	0.896	0.792	0.820	0.687	0.339	0.606
	WaveBeat (DBN)	0.864	0.711	**0.900**	0.748	0.563	**0.853**
	WaveBeat (BeatFCOS)	**0.927**	**0.873**	0.898	**0.807**	**0.697**	0.756
	Spectral TCN (Böck and Davies 2020)	0.962	0.947	0.961	0.916	0.913	0.960
	Hung et al. (Hung et al. 2022)	0.962	0.939	0.967	0.937	0.927	0.968
Hainsworth	WaveBeat (Peak)	0.755	0.609	0.662	0.466	0.182	0.388
	WaveBeat (DBN)	**0.778**	**0.712**	**0.829**	0.509	0.287	**0.643**
	WaveBeat (BeatFCOS)	0.761	0.678	0.735	**0.529**	**0.416**	0.500
	Spectral TCN (Böck and Davies 2020)	0.902	0.848	0.930	0.722	0.696	0.872
	Hung et al. (Hung et al. 2022)	0.877	0.862	0.915	0.748	0.738	0.870
Beatles	WaveBeat (Peak)	0.886	0.735	0.815	0.685	0.330	0.544
	WaveBeat (DBN)	0.893	0.786	**0.901**	0.758	0.473	**0.831**
	WaveBeat (BeatFCOS)	**0.903**	**0.797**	0.866	**0.762**	**0.579**	0.659
	Spectral TCN (Böck and Davies 2020)	–	–	–	0.837	0.742	0.862
	Hung et al. (Hung et al. 2022)	0.943	0.896	0.938	0.870	0.812	0.865
RWC Popular	WaveBeat (Peak)	0.836	0.681	0.755	0.646	0.336	0.483
	WaveBeat (DBN)	**0.864**	**0.771**	**0.905**	0.692	0.442	**0.793**
	WaveBeat (BeatFCOS)	0.862	0.763	0.849	**0.779**	**0.691**	0.731
	Spectral TCN (Böck and Davies 2020)	–	–	–	–	–	–
	Hung et al. (Hung et al. 2022)	0.950	0.925	0.958	0.945	0.939	0.959
GTZAN	WaveBeat (Peak)	0.809	0.644	0.723	0.520	0.175	0.458
	WaveBeat (DBN)	**0.831**	**0.716**	**0.847**	**0.567**	0.320	**0.730**
	WaveBeat (BeatFCOS)	0.808	0.682	0.773	0.546	**0.378**	0.543
	Spectral TCN (Böck and Davies 2020)	0.885	0.813	0.931	0.672	0.640	0.832
	Hung et al. (Hung et al. 2022)	0.887	0.812	0.920	0.756	0.715	0.881
SMC	WaveBeat (Peak)	0.413	0.167	0.250	–	–	–
	WaveBeat (DBN)	**0.431**	**0.288**	**0.431**	–	–	–
	WaveBeat (BeatFCOS)	0.400	0.244	0.315	–	–	–
	Spectral TCN* (Böck and Davies 2020)	0.544	0.443	0.635	–	–	–
	Hung et al.* (Hung et al. 2022)	0.605	0.514	0.663	–	–	–

Table 2.6 Results from WaveBeat using peak-picking, DBN, and BeatFCOS as well as the Spectral TCN (Böck and Davies 2020) mentioned in the original paper (Steinmetz and Reiss 2021) for comparison purposes. Results from the SOTA model by (Hung et al. 2022) by Hung et al. were included to allow for more comparison. The best scores pertaining to the WaveBeat were bolded, and all scores were reported using checkpoints trained with 8-fold validation. * indicates that this dataset was used during the training of this model.

3 Conclusion

Through the production of a model that mostly resembles object detection models yet performs the same task of beat and downbeat detection, we have created a bridge between the two research fields, unlocking a trove of lessons learned in the much more heavily researched field of object detection that can be easily adapted for use in beat tracking. We have also produced and tested a new, simpler type of post-processing that puts more reliance on the model itself to play the role of the DBN with the added classification and regression heads as well as adjacency constraint loss, which can be seen in itself as playing the role of the DBN, aligning beats and downbeats together. Due to its similarity in structure with object detection models that are designed to use other backbones, we can theoretically integrate BeatFCOS with any beat tracking model to train using beat intervals.

As WaveBeat was the first model to revisit the question regarding the use of DBNs, we naturally used it as a starting point for research. Although based on the results from Section 2.5.3 it is difficult to claim that BeatFCOS is superior to DBNs, we still see BeatFCOS as a very promising approach. One big reason for this is that DBNs are used to pick up beat and downbeat locations by applying criteria to the sequence, an incorporation of specific expertise regarding music theory and how beats and downbeats occur in each bar or measure. However, the usage of FPNs, linking of intervals to align sequentially with adjacency

constraints, and the three heads approach are much more general knowledge in comparison.

3.1 Future work

Integration with other models After beginning the research it was discovered through the conducting of more precise 8-fold cross validation that the performance for WaveBeat is a scale of magnitude lower than we had initially thought, and the cause of this is likely to be due to the fact that it is based on audio waveform instead of spectrograms. Thus, the effectiveness of the BeatFCOS paradigm when added to spectrogram-based models, continues to be up for question. Similarly to what we did with WaveBeat, integrating BeatFCOS with models such as the Spectral TCN (Böck and Davies 2020), Beat Transformer (Zhao et al. 2022), and the model by Hung et al. (Hung et al. 2022) will be a very interesting future experiment and a very clear next step.

Implementation of self-supervised learning In evaluating the model, we attempted training both with and without the pretrained WaveBeat backbone (Steinmetz and Reiss 2021). As pretrained ResNets (He et al. 2016) are generally used to help object detections converge, we also applied a similar principle, producing similarly positive results. In the future, instead of pretraining the backbone on labeled beat data, we can use an unsupervised learning approach by having the model learn general features of unlabeled music data, which are much more plentiful in comparison to labeled data. Similar approaches were made in natural language processing, resulting in stellar performance (Schneider et al. 2019). More recently we have seen similar efforts made in the area of pretrained music models. Among these efforts, the pretrained model

MusicBERT was trained on a corpus of one million songs (Zeng et al. 2021), and it is hard to imagine that training with labeled data on top of such a pretrained model would not produce better results, especially for held-out datasets like *GTZAN* (Tzanetakis and Cook 2002) and *SMC* (Holzapfel et al. 2012). It has also been noted by other works (Steinmetz and Reiss 2021)(MatthewDavies and Böck 2019) that the latter held-out dataset contains music with much more tricky beat and downbeat patterns in explaining why beat predictions come out notably lower, which is where self-supervised learning can be largely beneficial to.

Further optimization of the present model Due to time constraints, we were not able to test each part of the model as thoroughly as we wanted; many components in object detection were copied without intense thought on what sort of modifications or omissions can be made to produce an even better result for beat tracking. For example, we are still not fully confident that eight TCN blocks in the backbone, as well as the usage of just the top two blocks as inputs to the FPN, with it producing three more layers, is truly the most efficient approach, or if five pyramid layers is truly necessary to begin with. Although Soft-NMS is used due to the fact that it produced the best scores when performing testing, it is possible that the creation of a new version of NMS designed to work best with our problem would lead to even better results. There is also a possibility that the model is more large now than it has to be, and this needs to be experimented with as well.

Usage of more datasets The datasets used to train BeatFCOS with WaveBeat is also quite limited in comparison to the datasets used to train other

models and SMC, a dataset with difficult beat patterns, was noticeably only used by (Steinmetz and Reiss 2021) and us as a test dataset, which could have played a big factor to the ultimate performance of the model. The original WaveBeat paper mentions that some datasets used to train the Spectral TCN (Böck and Davies 2020) were skipped and compensated with the usage of data augmentation, but because audio waveform-based models are inherently data hungry, using more datasets to train both WaveBeat and BeatFCOS can potentially lead to even better test results.

Implementation of other improvements in object detection Since we have been able to bridge the gap between object detection and beat tracking with a model largely based on object detection models, now we are able to apply the same lessons and improvements in object detection to our model. We know this to be the case, as our original code was originally a fork a PyTorch implementation of RetinaNet (Lin et al. 2017b) but resulted in greater efficiency as subsequent lessons were applied, from the removal of anchor boxes (Tian et al. 2019), limiting of pixel positions that have the responsibility to predict (Tian et al. 2020), addition of group normalization on the heads (Wu and He 2018), usage of a 1D version of GIoU loss (Rezatofighi et al. 2019), and more. Such improvements were satisfyingly parallel to the respective improvements seen in counterpart object detection models. There are many effective, newer techniques in object detection that we have not had the chance to test out, including but not limited to Generalized Focal Loss (Li et al. 2020)(Li et al. 2021) and newer models such as VarifocalNet (Zhang et al. 2021). This is significant, as object detection is a much more heavily researched topic; we have demonstrated that the creation of this model allows many improvements

in object detection to simultaneously benefit the research of beat and downbeat tracking.

Bibliography

Bai, Shaojie, J Zico Kolter, and Vladlen Koltun, 2018: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Böck, Sebastian, and Matthew EP Davies, 2020: Deconstruct, analyse, reconstruct: how to improve tempo, beat, and downbeat estimation. *ISMIR*, 574–582.

Böck, Sebastian, Matthew EP Davies, and Peter Knees, 2019: Multi-task learning of tempo and beat: learning one to improve the other. *ISMIR*, 486–493.

Böck, Sebastian, Florian Krebs, and Gerhard Widmer, 2014: A multi-model approach to beat tracking considering heterogeneous music styles. *ISMIR*. Citeseer, 603–608.

Böck, Sebastian, Florian Krebs, and Gerhard Widmer, 2016: Joint beat and downbeat tracking with recurrent neural networks. *ISMIR*. New York City, 255–261.

Böck, Sebastian, and Markus Schedl, 2011: Enhanced beat tracking with context-aware neural networks. *Proc. Int. Conf. Digital Audio Effects*, 135–139.

Bodla, Navaneeth, Bharat Singh, Rama Chellappa, and Larry S Davis, 2017: Soft-nms–improving object detection with one line of code. *Proceedings of the IEEE international conference on computer vision*, 5561–5569.

Davies, Matthew EP, Norberto Degara, and Mark D Plumbley, 2009: Evaluation methods for musical audio beat tracking algorithms. *Queen Mary University of London, Centre for Digital Music, Tech. Rep. C4DM-TR-09-06*.

De Clercq, Trevor, and David Temperley, 2011: A corpus analysis of rock harmony. *Popular Music*, **30**, 47–70.

Doucet, Arnaud, Nando De Freitas, and Neil Gordon, 2001: An introduction to sequential monte carlo methods. *Sequential Monte Carlo methods in practice*. Springer, 3–14.

Eyben, Florian, Sebastian Böck, Björn Schuller, and Alex Graves, 2010: Universal onset detection with bidirectional long-short term memory neural networks. *Proc. 11th Intern. Soc. for Music Information Retrieval Conference, ISMIR, Utrecht, The Netherlands*, 589–594.

Gopali, Saroj, Faranak Abri, Sima Siami-Namini, and Akbar Siami Namin, 2021: A comparative study of detecting anomalies in time series data using lstm and tcn models. *arXiv preprint arXiv:2112.09293*.

Goto, Masataka, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka, 2002: Rwc music database: popular, classical and jazz music databases. *Ismir*. Volume 2, 287–288.

Goto, Masataka, and Yoichi Muraoka, 1994: A beat tracking system for acoustic signals of music. *Proceedings of the second ACM international conference on Multimedia*, 365–372.

Gouyon, Fabien, 2006: *A computational approach to rhythm description-Audio features for the computation of rhythm periodicity functions and their use in tempo induction and music content processing*. Universitat Pompeu Fabra.

Gouyon, Fabien, Anssi Klapuri, Simon Dixon, Miguel Alonso, George Tzanetakis, Christian Uhle, and Pedro Cano, 2006: An experimental comparison of audio tempo induction algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, **14**, 1832–1844.

Hainsworth, Stephen W, and Malcolm D Macleod, 2004: Particle filtering applied to musical tempo tracking. *EURASIP Journal on Advances in Signal Processing*, **2004**, 1–11.

Hargreaves, David J et al., 1986: *The developmental psychology of music*. Cambridge University Press.

Harte, Christopher, 2010: "Towards automatic extraction of harmony information from music signals". PhD thesis.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 2015: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 2016: Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Heydari, Mojtaba, Frank Cwitkowitz, and Zhiyao Duan, 2021: Beatnet: crnn and particle filtering for online joint beat downbeat and meter tracking. *arXiv preprint arXiv:2108.03576*.

Holzapfel, Andre, Matthew EP Davies, José R Zapata, João Lobato Oliveira, and Fabien Gouyon, 2012: Selective sampling for beat tracking evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, **20**, 2539–2548.

Holzapfel, Andre, Florian Krebs, and Ajay Srinivasamurthy, 2014: Tracking the "odd": meter inference in a culturally diverse music corpus. *ISMIR-International Conference on Music Information Retrieval*. ISMIR, 425–430.

Huang, Jonathan, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al., 2017: Speed/accuracy trade-offs for modern convolutional object detectors. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7310–7311.

Huang, Lichao, Yi Yang, Yafeng Deng, and Yinan Yu, 2015: Densebox: unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*.

Hung, Yun-Ning, Ju-Chiang Wang, Xuchen Song, Wei-Tsung Lu, and Minz Won, 2022: Modeling beats and downbeats with a time-frequency transformer. *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 401–405.

Ioffe, Sergey, and Christian Szegedy, 2015: Batch normalization: accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*. PMLR, 448–456.

Krebs, Florian, Sebastian Böck, Matthias Dorfer, and Gerhard Widmer, 2016: Downbeat tracking using beat synchronous features with recurrent neural networks. *ISMIR*, 129–135.

Krebs, Florian, Sebastian Böck, and Gerhard Widmer, 2013: Rhythmic pattern modeling for beat and downbeat tracking in musical audio. *Ismir*. Citeseer, 227–232.

Krebs, Florian, Andre Holzapfel, Ali Taylan Cemgil, and Gerhard Widmer, 2015: Inferring metrical structure in music using particle filters. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, **23**, 817–827.

Law, Hei, and Jia Deng, 2018: Cornernet: detecting objects as paired keypoints. *Proceedings of the European conference on computer vision (ECCV)*, 734–750.

Li, Xiang, Wenhai Wang, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang, 2021: Generalized focal loss v2: learning reliable localization quality estimation for dense object detection. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11632–11641.

Li, Xiang, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang, 2020: Generalized focal loss: learning qualified and distributed bounding boxes for dense object detection. *Advances in Neural Information Processing Systems*, **33**, 21002–21012.

Lin, Tsung-Yi, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie, 2017a: Feature pyramid networks for object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2117–2125.

Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár, 2017b: Focal loss for dense object detection. *Proceedings of the IEEE international conference on computer vision*, 2980–2988.

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, 2014: Microsoft coco: common objects in context. *European conference on computer vision*. Springer, 740–755.

Lu, Wei-Tsung, Ju-Chiang Wang, Minz Won, Keunwoo Choi, and Xuchen Song, 2021: Spectnt: a time-frequency transformer for music audio. *arXiv preprint arXiv:2110.09127*.

Marchand, Ugo, and Geoffroy Peeters, 2015: Swing ratio estimation. *Digital Audio Effects 2015 (Dafx15)*.

Matthew E. P. Davies Sebastian B ock, Magdalena Fuentes, Nov. 2021: *Tempo, Beat and Downbeat Estimation*. https://tempobeatdownbeat.github.io/tutorial/intro.html. URL: `https://tempobeatdownbeat.github.io/tutorial/intro.html`.

MatthewDavies, EP, and Sebastian Böck, 2019: Temporal convolutional networks for musical audio beat tracking. *2019 27th European Signal Processing Conference (EUSIPCO)*. IEEE, 1–5.

McFee, Brian, Eric J Humphrey, and Juan Pablo Bello, 2015: A software framework for musical data augmentation. *ISMIR*. Volume 2015, 248–254.

Murphy, Kevin Patrick, 2002: *Dynamic bayesian networks: representation, inference and learning*. University of California, Berkeley.

Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu, 2016: Wavenet: a generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

O'Shea, Keiron, and Ryan Nash, 2015: An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Peeters, Geoffroy, and Helene Papadopoulos, 2010: Simultaneous beat and downbeat-tracking using a probabilistic framework: theory and large-scale evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, **19**, 1754–1769.

Rabiner, Lawrence R, 1989: A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, **77**, 257–286.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun, 2015: Faster r-cnn: towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, **28**.

Rezatofighi, Hamid, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese, 2019: Generalized intersection over union: a metric and a loss for bounding box regression. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 658–666.

Rouzic, Michel, 2009: *Analysis & resynthesis sound spectrograph*.

Roy, William G, and Timothy J Dowd, 2010: What is sociological about music? *Annual Review of Sociology*, **36**, 183–203.

Schlüter, Jan, and Thomas Grill, 2015: Exploring data augmentation for improved singing voice detection with neural networks. *ISMIR*, 121–126.

Schneider, Steffen, Alexei Baevski, Ronan Collobert, and Michael Auli, 2019: Wav2vec: unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*.

Srinivasamurthy, Ajay, Andre Holzapfel, Ali Taylan Cemgil, and Xavier Serra, 2015: Particle filters for efficient meter tracking with dynamic bayesian networks. *Müller M, Wiering F, editors. ISMIR 2015. 16th International Society for Music Information Retrieval Conference; 2015 Oct 26-30; Málaga, Spain. Canada: ISMIR; 2015.* International Society for Music Information Retrieval (ISMIR).

Srinivasamurthy, Ajay, Andre Holzapfel, Ali Taylan Cemgil, and Xavier Serra, 2016: A generalized bayesian model for tracking long metrical cycles in acoustic music signals. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 76–80.

Srinivasamurthy, Ajay, and Xavier Serra, 2014: A supervised approach to hierarchical metrical cycle tracking from audio music recordings. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5217–5221.

Steinmetz, Christian J, and Joshua D Reiss, 2021: Wavebeat: end-to-end beat and downbeat tracking in the time domain. *arXiv preprint arXiv:2110.01436*.

Tian, Yuandong, Xinlei Chen, and Surya Ganguli, 2021: Understanding self-supervised learning dynamics without contrastive pairs. *International Conference on Machine Learning*. PMLR, 10268–10278.

Tian, Zhi, Chunhua Shen, Hao Chen, and Tong He, 2019: Fcos: fully convolutional one-stage object detection. *Proceedings of the IEEE/CVF international conference on computer vision*, 9627–9636.

Tian, Zhi, Chunhua Shen, Hao Chen, and Tong He, 2020: Fcos: a simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Tzanetakis, George, and Perry Cook, 2002: Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, **10**, 293–302.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, 2017: Attention is all you need. *Advances in neural information processing systems*, **30**.

Wang, Sun-Chong, 2003: Artificial neural network. *Interdisciplinary computing in java programming*. Springer, 81–100.

Whiteley, Nick, A Taylan Cemgil, and Simon Godsill: Bayesian modelling of temporal structure in musical audio ().

Whiteley, Nick, A Taylan Cemgil, and Simon Godsill, 2007: Sequential inference of rhythmic structure in musical audio. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*. Volume 4. IEEE, IV–1321.

Wu, Yuxin, and Kaiming He, 2018: Group normalization. *Proceedings of the European conference on computer vision (ECCV)*, 3–19.

Yu, Fisher, and Vladlen Koltun, 2015: Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*.

Yu, Jiahui, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang, 2016: Unitbox: an advanced object detection network. *Proceedings of the 24th ACM international conference on Multimedia*, 516–520.

Zeng, Mingliang, Xu Tan, Rui Wang, Zeqian Ju, Tao Qin, and Tie-Yan Liu, 2021: Musicbert: symbolic music understanding with large-scale pre-training. *arXiv preprint arXiv:2106.05630*.

Zhang, Haoyang, Ying Wang, Feras Dayoub, and Niko Sunderhauf, 2021: Varifocalnet: an iou-aware dense object detector. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8514–8523.

Zhao, Jingwei, Gus Xia, and Ye Wang, 2022: Beat transformer: demixed beat and downbeat tracking with dilated self-attention. *arXiv preprint arXiv:2209.07140*.

국 문 초 록

비트 및 다운비트 추적은 음악 신호 처리의 기본 작업이며 음악 분석 및 생성에 수많은 응용 프로그램이 있지만 대부분의 신경망은 예측된 비트의 최종 집합을 추출하기 위해 동적 베이지안 네트워크(DBN)의 사용에 의존한다. 본 논문은 객체 탐지에서 일반적으로 사용되는 구성 요소를 활용하면서 비트 및 다운비트 추적을 위한 DBN이 없는 접근 방식을 제안한다. FCOS 객체감지 모델은 WaveBeat 비트 추적 모델을 백본(backbone)으로 사용하여 1D 음악 오디오를 입력으로 받아들이도록 수정하며, 계층적 특징을 포착하기 위해 기능 피라미드 네트워크(FPN)와 통합한다. 한 가지 중요한 기여는 전통적으로 외부 수정에 사용되는 DBN을 사용하여 사후 처리할 필요가 없다는 것이다. 이는 모델에게 비트 시점 뿐만 아니라 각 비트 쌍 사이의 간격 길이를 학습하여 최종 비트 집합을 생성하도록 학습시키는 것으로 수행된다. 이 접근 방식은 WaveBeat에서 사용하는 동일한 음악 데이터 세트와 비교되며, DBN을 사용하는 WaveBeat과 일치할 때 경쟁력 있는 결과를 보여준다. 이 연구는 또한 큰 변화 없이 음악 오디오에서 비트 및 다운비트 추적을 위해 객체탐지 기술을 사용할 수 있는 것을 보여준다.

주제어(키워드, 색인어): 비트 추적, 다운비트 추적, 음악정보검색, 객체탐지, 신경망, 딥러닝, 인공지능