

Aapo Hakala

# SONG IDENTIFICATION FROM LIVE MUSIC USING SIAMESE CONVOLUTIONAL NEURAL NETWORKS

# Abstract

Aapo Hakala: SONG IDENTIFICATION FROM LIVE MUSIC USING SIAMESE CONVOLUTIONAL NEURAL NETWORKS

Master's thesis

Tampere University

Master's Degree Programme in Electrical Engineering

May 2023

---

The task of live music song identification is about identifying the underlying song being performed in a live recording track. Automated song identification systems have long been used in applications like Shazam and Youtube content-id as tools for music information retrieval and copyright infringement detection, respectively. However, the fingerprinting-based methods used in audio copy detection perform rather poorly when musical changes are concerned. Two recordings presenting the same song can have differences in their tempo, song key, instrumentation, arrangement, chords, melodies, rhythms and lyrics. Live music environment brings additional complexity to the task as crowd noises, improvisation of performing artists and inaccuracies in timing or pitch of played notes can further alter the song. Recent studies about cover song identification have managed to develop deep learning methods that are robust to musical changes. However, it is unknown how well these methods work with live music due to the lack of ongoing research about the topic.

This study aims to solve the problem of live music song identification by applying modern deep learning methods in an information retrieval system that, given a live recording of a song, is able to retrieve the original version of the song from a music database. Supervised similarity learning is used in the development of a siamese convolutional neural network-based model. The model makes use of cross-similarity matrices of multi-level deep sequences in measuring the musical similarity between different audio tracks. The performance of the system and its robustness to musical changes are tested with a manually collected custom live music dataset. The results of the experiments show that the proposed system is able to identify 81,1% of the live recordings in the collected dataset.

**Keywords:** song identification, live music, similarity learning, siamese convolutional neural network, multi-level deep sequences, information retrieval.

The originality of this thesis has been checked using the Turnitin Originality Check service.

# Contents

|       |  |    |
|-------|--|----|
| 1     | Introduction . . . . .                                   | 1  |
| 2     | Background . . . . .                                     | 3  |
| 2.1   | Information Retrieval . . . . .                          | 3  |
| 2.1.1 | Evaluation Metrics . . . . .                             | 3  |
| 2.2   | Similarity Learning . . . . .                            | 4  |
| 2.3   | Convolutional Neural Networks . . . . .                  | 5  |
| 2.3.1 | Convolutional Layer . . . . .                            | 6  |
| 2.3.2 | 1D-Convolution . . . . .                                 | 7  |
| 2.3.3 | 2D-Convolution . . . . .                                 | 9  |
| 2.3.4 | Padding and Stride . . . . .                             | 10 |
| 2.3.5 | Pooling layer . . . . .                                  | 12 |
| 2.3.6 | Fully connected layer . . . . .                          | 13 |
| 2.4   | Siamese Convolutional Neural Networks . . . . .          | 14 |
| 2.5   | Training Siamese Convolutional Neural Networks . . . . . | 16 |
| 2.5.1 | Loss Function . . . . .                                  | 16 |
| 2.5.2 | Backpropagation . . . . .                                | 17 |
| 2.5.3 | Gradient descent . . . . .                               | 19 |
| 2.6   | Frequency Domain Presentation of a Signal . . . . .      | 20 |
| 2.6.1 | Short-time Fourier Transform . . . . .                   | 20 |
| 2.6.2 | Constant Q-Transform . . . . .                           | 23 |
| 2.7   | Cross-similarity Matrix . . . . .                        | 24 |
| 3     | Live Music Song Identification System . . . . .          | 26 |
| 3.1   | System Overview . . . . .                                | 26 |
| 3.2   | Feature Extraction . . . . .                             | 27 |
| 3.3   | Feature Representation . . . . .                         | 28 |
| 3.4   | Cross Similarity Matrices . . . . .                      | 28 |
| 3.5   | Similarity Measuring . . . . .                           | 29 |
| 4     | Experiments . . . . .                                    | 31 |
| 4.1   | Data . . . . .   | 31 |
| 4.2   | Model Parametrization . . . . .                          | 32 |
| 4.3   | Experiment Results . . . . .                             | 33 |
| 5     | Discussion . . . . .                                     | 34 |
| 5.1   | Feature Analysis . . . . .                               | 35 |
| 5.2   | Future improvements . . . . .                            | 37 |
| 6     | Conclusions . . . . .                                    | 39 |

|  |    |
|--|----|
| References . . . . .                         | 43 |
| APPENDIX Custom Live Music Dataset . . . . . | 44 |

# 1 Introduction

Humans can easily recognize an alternative version of a familiar song even if they have drastic musical and structural differences. Two versions of the same song can have differences in their instrumentation, arrangement, tempo, lyrics, song key, individual chords or entire chord progressions and even changes in song defining melodies. Live music environment complicates the recognition task even further as crowd noises and improvisation of performing artists are likely to occur. Automatic live music song identification would be highly beneficial to songwriters, publishers and performance rights societies, who must currently make reports of live performances manually in order to collect performance rights royalties. Further automation of live performance tracking would also make it easier for smaller event organizers to report about live music that has been played. Consequently, there would be more royalty incomes for those who rightfully deserve a full compensation for their creative work.

Automatic song identification systems have been developed and used extensively ever since the widespread use of smart phones. *Acoustic fingerprinting* used to be the state of the art method for automatic song recognition and it is still used in applications like Youtube Content-ID and Shazam for preventing plagiarism and to provide information about songs played in the background [13][35]. These systems have been proven to perform well in *audio copy detection* while also being robust to background noises. The problem with acoustic fingerprinting is that it can not deal with considerable musical changes, which is why the fingerprinting methods have not made a similar breakthrough in live music applications or in the field of *cover song identification* (CSI). Although Youtube Content-ID is known to be able to detect cover versions from uploaded videos to some extent, it is not common knowledge whether there are more advanced methods being used in addition to acoustic fingerprinting [3].

The problem setting of live music song identification is very similar to that of CSI. In both tasks the goal is to find out whether two musical recordings are derived from the same musical composition. In live music song identification the performing artists can be the same in both recordings unlike in CSI. Another difference is that in live music song identification one of the two recordings is an original studio version of a song while the other is a live recording of a song. The problem can be approached as an *information retrieval* (IR) task, where queries of live music recordings are given to an IR-system that needs to retrieve the corresponding studio recordings from a database of songs.

Recently the state-of-the-art methods in the field of CSI have been systematically

applying modern deep learning techniques, namely *convolutional neural networks* (CNNs) [6][37][7][8]. These studies have made it clear that deep neural network-based solutions outperform conventional methods relying on various combinations of feature extraction and distance metrics [5][30][31]. But when it comes to live music song identification, none of the CNN-based methods provide answers about how they perform with live music data. So far there has only been attempts to solve the problem using conventional methods [15][34][26]. Given the superiority of modern deep learning techniques in CSI, it is reasonable to assume that live music song identification should be also approached by using the latest state-of-the-art methods.

This thesis presents a method for live music song identification system that is based on *similarity learning* and *siamese convolutional neural networks* (SCNN). The method applies *Cross-similarity Matrices* (CSMs) of multi-level deep sequences as proposed by Jiang et al. in their related work[16] for conventional CSI. A custom Live Music Dataset was collected manually to evaluate the performance of the implemented model with live music data.

The structure of the thesis is the following. Chapter 2 provides theoretical background about all the relevant topics related to deep learning and live music song identification. A detailed system description about the proposed method as well as the model architecture are then presented in Chapter 3. Information about the data and the experiments are presented in Chapter 4 along with the results of the experiments. Finally, observations and conclusions about the system performance are discussed in chapters 5 and 6.

## 2 Background

Automatic song identification from live music includes multiple concepts from information science to advanced signal processing and machine learning methods. The second chapter aims to break down these constituents by presenting the theoretical background of all the topics being applied in the system. Brief introductions to IR and *similarity learning* are first provided in sections 2.1 and 2.2, respectively. A more detailed discussion about CNNs and SCNNs are then given in sections 2.3, 2.4 and 2.5. The basics of frequency domain presentations are covered in Section 2.6.1. Finally, the fundamentals of cross-similarity matrices are discussed in Section 2.7.

### 2.1 Information Retrieval

IR in its classical form consists of the following elements: A need for information, a collection of resources and an IR system connecting the two [1]. The need for information is first presented as a query to an IR system. The query is given by a user and it describes the desired information, often in an inexact way. The IR system processes the query and aims to retrieve documents from the collection of resources that are relevant to the need of information. Examples of applications performing IR include search engines, digital libraries, recommendation systems and instances of media searching such as music and image retrieval.

The performance of an IR system depends on how well it succeeds in finding and retrieving the relevant items from all the available resources. In practice the evaluation is not a straightforward task as there are several different scenarios to be taken into account. For instance, when there are multiple relevant documents matching a query the system may only retrieve a subset of all the relevant items. On the other hand the system may retrieve more documents than what are truly relevant, thus falsely assuming some of the non-relevant items to be relevant. Having an IR system with a relevance-based ranking allows numerical comparison of resources and furthermore enables evaluating the overall performance of the system in a consistent way. Relevance-based ranking means that for each query the collection of resources are sorted according to a desired scoring mechanism. Documents yielding a high relevance-score are considered more relevant than low-scoring items.

#### 2.1.1 Evaluation Metrics

Multiple evaluation metrics are needed in order measure and describe the overall performance of an IR system. The following metrics have been commonly used in the field of CSI and are therefore also expected to be useful in the task of live music

detection [22]. The used evaluation metrics are *precision at 10* (P@10), *the mean rank of the first correctly identified cover* (MR1) and *mean average precision* (MAP). A brief description about each metric is presented below.

P@10 quantifies the proportion of relevant items among the ten highest ranked results retrieved by an IR system. When  $q$  is a query,  $Q$  is the total number of queries and  $r_q$  is the number of relevant items in the top-10 results for a query, the formula for calculating P@10 over a set of queries is written as:

$$P@10 = \frac{1}{Q} \sum_{q=1}^Q \frac{r_q}{10}. \quad (2.1)$$

MR1 is a metric for measuring the rank of the best relevant item. It considers only one of many possible relevant items retrieved by an IR system. Let  $Q$  be the total number of queries and let  $r_1$  present the rank of the first relevant item of a query  $q$ . MR1 is then calculated as:

$$MR1 = \frac{1}{Q} \sum_{q=1}^Q r_1. \quad (2.2)$$

In order to present MAP, *Average Precision* (AP) needs to be calculated first. AP is a metric that evaluates whether all of the ground-truth relevant items retrieved by the system are ranked higher or not. It considers all the relevant items and its equation is calculated as:

$$AP = \frac{\sum_r P@r}{R}, \quad (2.3)$$

where  $r$  denotes the rank of each relevant item,  $R$  is the total number of relevant items, and P@r is the precision of the top-r retrieved documents. MAP can then be obtained by taking the arithmetic mean of the AP values over  $Q$  queries. It is expressed as

$$MAP = \frac{1}{Q} \sum_{q=1}^Q AP_q. \quad (2.4)$$

## 2.2 Similarity Learning

Similarity learning is an area of supervised learning that focuses on studying the similarity of objects. Similarity learning is often used to solve problems where there are no fixed number of classes that the examined objects fit into. Face verification and signature verification are examples of such tasks where similarity learning has been applied recently [36][21]. The goal in similarity learning is to learn a *similarity function* that measures the similarity between the objects being compared. The similarity function produces a *similarity score* that describes the objects' similarity

as a numeric value. A high similarity score should be produced when the objects are similar to each other whereas dissimilar objects should yield a lower score.

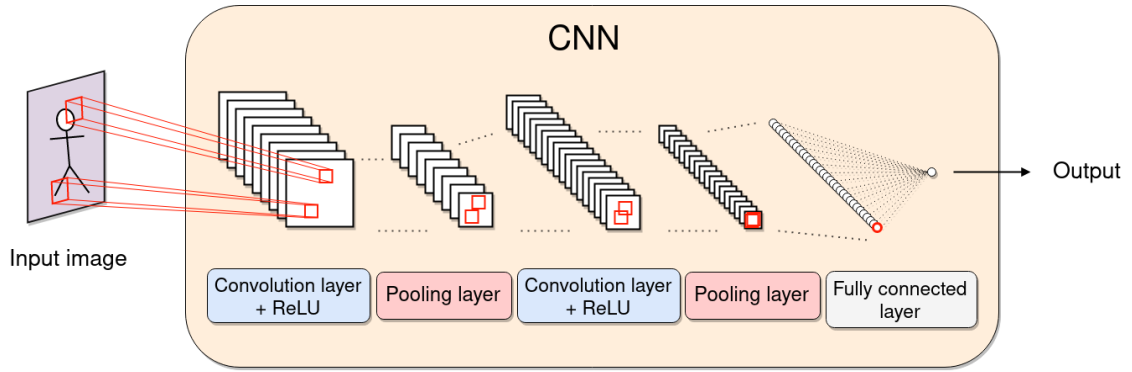
The topic of *distance metric learning* is closely related to similarity learning. In distance metric learning the goal is to learn a metric that assigns a small distance to pairs of objects that are semantically similar. Examples of traditional approaches applying distance metric learning include *principle component analysis*, *linear discriminant analysis* and *neighbourhood component analysis*. In distance metric learning the similarity function must obey the four axioms of a distance metric that are non-negativity, identity of indiscernibles, symmetry and triangle inequality. When using deep neural networks for similarity learning, not all four axioms of a distance metric are necessarily true and the network learns a so called *pseudometric* instead.

The main motivation for using deep neural networks in similarity learning is that they enable the comparison of more complex objects compared to traditional methods. Consequently, it is often required to use a simplified yet a meaningful representation of the input data called *embeddings*. An embedding is lower-dimensional vector presentation of the input data that maps the input features into an *embedding space*. The distances between different objects in the embedding space corresponds to their semantic similarity. Therefore, simple metrics such as the Euclidean distance can be used in the embedding space to compare the similarity of different objects.

## 2.3 Convolutional Neural Networks

CNN is a type of artificial neural network that is commonly used in the fields of image recognition and computer vision. A large variety of applications from medical imaging to autonomous cars and natural language processing are using CNNs for solving tasks of image recognition, classification and segmentation [2][32][12]. CNNs are capable of effectively recognizing patterns from two dimensional data due to its clever design which is comparable to the neural connections in animal visual cortex [14]. The history of CNNs dates back to early 80's when the first version of a CNN-like network architecture was proposed [10]. The real breakthrough of CNNs happened several decades later after traditional neural networks had become common and the high processing power of modern computers became widely available.

A simple design of a CNN architecture is visualized in Figure 2.1. The input of a CNN usually consists of *convolution layers*, *activation functions* and *pooling layers* that are stacked one after another in repetitive sequences. Any combination of these layers form the feature extraction part of the network that are also known as *hidden layers*. The output of the CNN contains a *Fully connected network* with one or more output neurons followed by a final *activation layer*. The part of the network between the fully connected layer and the output is responsible for performing the



**Figure 2.1** A block diagram of a generic CNN architecture.

classification according to high-level features that are extracted in the hidden layers.

In order to understand how CNNs work in detail it is necessary to familiarize oneself with all the different layers types and the mathematics behind them. Convolutional layers are first discussed in Section 2.3.1. An introduction to 1D- and 2D-convolution is then provided in corresponding sections 2.3.2 and 2.3.3. The concepts of padding and stride are explained in Section 2.3.4 followed by the topics of pooling and fully connected layers in sections 2.3.5 and 2.3.6, respectively.

### 2.3.1 Convolutional Layer

The purpose of convolutional layers is to learn to emphasize or hide certain characteristics in the given inputs. The way in which convolutional layers process data distinguishes CNNs from other types of neural networks. Instead of computing general matrix multiplication, convolutional layers perform a series of filtering operations that make use of the mathematical operation of *convolution*.

Convolutional layers have a grid-like topology of neurons that are arranged in three dimensions. The width and height axes of the grid correspond to those of the input image, while the depth of the layer determines the number of *channels* and the number of filters being used. A single filter consists of a small matrix of weights and a bias term. Each channel has its own filter meaning that all neurons at the same channel are using the same filter when computing their output. During the training of the model the weights and biases in each filter are tuned to detect particular features and data patterns from the input channels, such as edges or curved shapes. The outputs produced by a convolution layer are abstracted versions of the input data and they are commonly referred as *feature maps* or *activation maps*. The produced feature maps contain information about the presence of the data patterns of interest as neuron activations.

All neurons receive their inputs from a restricted and spatially local area of the previous layer. This is known as *local connectivity* of CNNs, meaning that the spatial

surroundings of each pixel or neuron are always taken into account when passing data from one layer to another. A typical *receptive field* of a neuron overlaps with the neighbouring neurons receptive fields and has a rectangular shape. An uneven number of pixels, such as  $3 \times 3$  or  $5 \times 5$  are often preferred over even ones, but larger segments can also be used with different shapes [20]. In convolutional layers the size and shape of the receptive field of a neuron is equal to the dimensions of the filters being used.

By stacking convolutional layers one after another the receptive field of the neurons gets larger and larger the deeper in the network they are located. In other words, the early layers of a CNN creates representations of only small regions of the input, which then become increasingly overlapping and global towards the end of the network. An illustration of gradually increasing receptive fields can be seen in figure 2.1.

Convolutional layers are often followed by some activation function such as *Rectified Linear Unit* (ReLU). ReLU-activation is used to introduce non-linearities to the network and to speed up the training procedure of the network [18]. Other activation functions like *hyperbolic tangent* or *sigmoid* are known to be less effective between hidden layers in terms of learning performance [11]. However, depending on the task they might come handy in the final layers of a CNN when mapping the network output into a specific numerical range.

### 2.3.2 1D-Convolution

Convolution is a mathematical operation between two functions where one of them is used to filter the other. More specifically, convolution is the integral of the product of the two functions after reversing and shifting one of the functions. Let  $f(t)$  and  $g(t)$  be continuous input functions and let  $*$  denote the convolution operation. The definition of convolution can now be written as:

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau, \quad (2.5)$$

where  $\tau$  is a *dummy* or a *bound variable* used for the intermediate computation. Discrete convolution is derived similarly for discontinuous signals. When  $f, g \in \mathbb{Z}$  are discrete functions sampled by  $n$  elements, the discrete version of convolution is then given by:

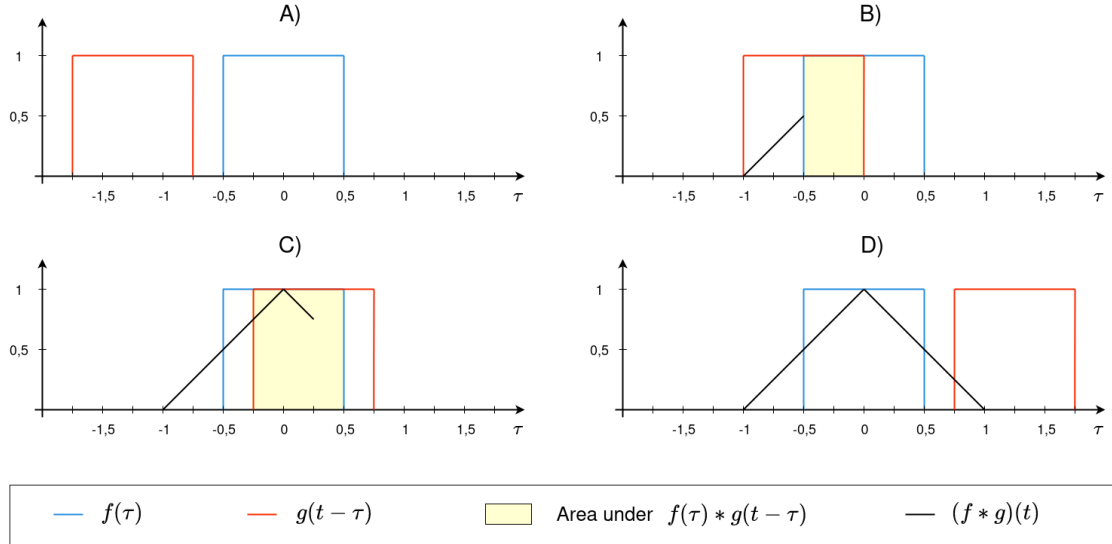
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m]. \quad (2.6)$$

In practice it is not possible to process infinitely long data sequences so a finite version is needed. When dealing with finite signals, Equation 2.6 can be written

as:

$$(f * g)[n] = \sum_{m=-M}^M f[m]g[n - m], \quad (2.7)$$

where  $M$  is the number of discrete samples forming the function  $g$ . Equations 2.5, 2.6 and 2.7 are easiest to comprehend with a visual presentation as shown in Figure 2.2.



**Figure 2.2** Illustration of a convolution operation of identical input functions  $f(t)$  and  $g(t)$  divided in four panels A, B, C and D.

The example in Figure 2.2 is divided in four panels A, B, C and D, each presenting a snapshot of the convolution process in a chronological order. Let the two input functions  $f(t) = g(t)$  be rectangular functions with a height and width of 1. At first,  $g(t - \tau)$  is obtained by mirroring the input function  $g(t)$  horizontally. Hence the negative term  $-\tau$  in the equation. The function used as a filter is commonly referred as *the kernel*. The integral of the product of the two functions from negative infinity to positive infinity effectively means that  $g(t - \tau)$  slides along the  $\tau$ -axis from left to right, meanwhile  $f(\tau)$  remains stationary.

In panel A of Figure 2.2 the red rectangle presenting  $g(t - \tau)$  is being shifted towards the blue kernel function  $f(\tau)$ . At this point the product of the two functions as well as the resulting integral is zero. The output will remain zero until the sliding function starts to overlap with the kernel function after a certain amount of shift  $t$ . In panel B the two functions are now overlapping and thus the resulting integral gets a positive value as shown by the black line. Note how the output of the integral is constantly increasing as the yellow intersection area between  $f(\tau)$  and  $g(t - \tau)$  gets bigger. In panel C the sliding function has now passed the point of maximum overlapping ( $\tau = 0$ ). The integral value starts to decrease compared to previous values since the intersection area is now decreasing. Finally, in panel D the

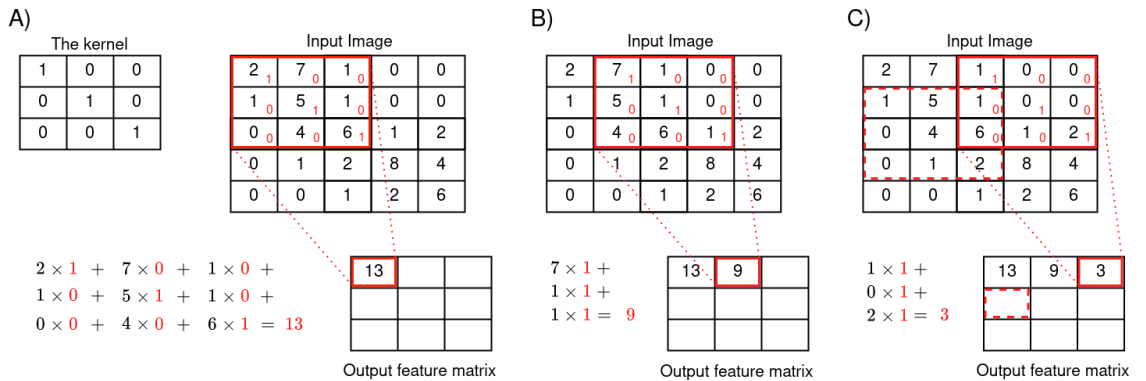
functions  $f(\tau)$  and  $g(t - \tau)$  are no longer overlapping, so the integral gets to zero and remains there as the sliding function approaches to infinity. The final result of the convolution is a curve with a shape of an isosceles triangle that is drawn in panel D.

### 2.3.3 2D-Convolution

The process of convolution can be performed on multi-dimensional signals as well. Digital images are essentially two dimensional matrices whose visual content is created by a grid of individual pixel values forming the matrix. Like 1D convolution, 2D convolution is also done by multiplying and accumulating the values between the overlapping parts of the two operands. 2D convolution can be viewed as an extension of 1D convolution that now consists of horizontal and vertical directions in two dimensional spatial domain. The definition of 2D convolution with discrete data is written as:

$$y[m, n] = f[m, n] * g[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} f[i - j]g[m - i, n - j], \quad (2.8)$$

where  $i$  and  $j$  specify the elements of the matrices  $f$  and  $g$ . A simple example of 2D convolution is shown in Figure 2.3.



**Figure 2.3** An example of a  $5 \times 5$ -image being convolved with a  $3 \times 3$ -kernel. The three stages A, B and C show how the sliding kernel is used to make element-wise calculations. For each position of the kernel the corresponding result is stored to an output feature matrix (i.e. feature map) while preserving the spatial context of the input image.

An example of 2D convolution is shown in Figure 2.3. In convolution layers performing 2D convolution the kernel is a small matrix of weights where each element is a trainable network parameter. For simplicity, an identity matrix with a size of  $3 \times 3$  is used as the kernel in the example and the bias term has been excluded. The input image being convolved is a  $5 \times 5$ -matrix containing random integers as its pixel values. In stage A the kernel is first placed on the top left corner of the

input image as highlighted in red. The sum of element-wise multiplications is then calculated between the aligned pixels of the input image and the kernel. A single output value is produced as the result which is stored into the output feature matrix as the top left pixel value. Note that the sum of multiplications can be alternatively referred as a dot product or a weighted sum.

In stage B of Figure 2.3 the sliding kernel has moved one step forward along the top three rows of the input image. The aforementioned calculations are repeated for the slightly different part of the input image. The resulting output value is then stored next to the previous result of the output feature map. In stage C the kernel has moved another step forward and the dot product between the kernel and the input image is obtained for the top right corner of the image. Now the sliding kernel has reached the edge of the image so the kernel is then relocated at the beginning of the next row as illustrated by the dashed rectangles. The same process is continued until the kernel has gone through every possible location on the input image and the resulting output feature matrix is fully produced.

The previous example shows how the spatial context of individual pixels from the input image is mostly preserved in the produced output feature matrix. This is because of the sliding kernel mechanism that enables focusing on small regions of the input image at a time while storing the results roughly at the same location in the output feature matrix. It is this property that makes 2D convolution more efficient compared to a fully connected layer structure in terms of required number of trainable parameters. For instance, when using a fully connected layer for the task in Figure 2.3 it would require  $25 \times 9 = 225$  weight parameters as each pixel from the input image would need to be connected with every element of the output feature matrix. With 2D convolution there are only 9 trainable weight parameters when using a  $3 \times 3$ -kernel whose weights remain the same regardless of the position of the sliding kernel.

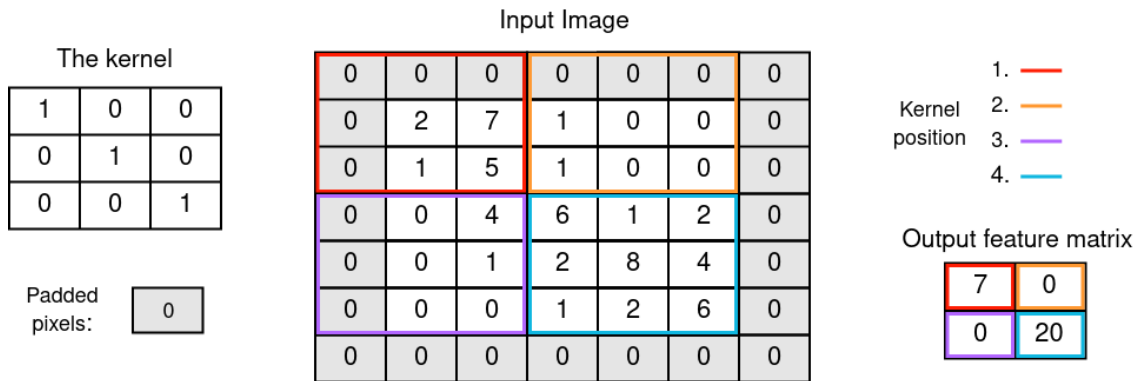
### 2.3.4 Padding and Stride

The previous example of 2D convolution resulted in an output feature matrix whose size was smaller than the original input image. This is because the central pixel of the sliding kernel gets never aligned with the pixels at the edge of the input image. In other words, there are no pixels for the kernel to extend to outside the image.

The technique of *padding* can be used to solve this problem. With padding one can have the same sized images before and after the convolution. In padding the input image is extended with rows and columns of additional pixels that are generated on each side of the image. In zero-padding, as the name implies, zero valued pixels are used to enlarge the image. This way the generated pixels have no effect on the weighted sums as the kernel weights are multiplied by the padded

zeros. At the same time the original edge pixels are allowed to align with the center of the kernel causing the output feature matrix to preserve the original input size.

*Striding* is another useful technique that can be used when convolving an image. Striding has the effect of purposely reducing the size of the produced output feature matrix. A stride determines how many pixels the sliding kernel will skip between each step during the convolution. An example of both zero-padding and striding are shown in Figure 2.4.



**Figure 2.4** An example of 2D convolution with zero-padding and striding. A stride of  $(3,3)$  is being used. Padded pixels on the sides of the original image are marked in gray. The positions of the sliding kernel at each step are shown in red, orange, purple and light blue rectangles in the corresponding order.

The example of zero-padding and a striding in Figure 2.4 has the same input image and kernel as the example in Figure 2.3. The zero-padded pixels around the input image have been highlighted in gray to illustrate the effect of padding. The padded image is treated as any  $6 \times 6$ -image and convolved normally by first placing the kernel on the top left corner of the enlarged image. The initial placement of the kernel is again outlined with a red square.

Striding is taken into account whenever sliding the kernel between each step. A stride of  $(3, 3)$  means that the kernel is moved three steps forwards when sliding the kernel horizontally, and three steps downwards when changing the kernel location from one row to another. In Figure 2.4, after computing the weighted sum for the top left corner the kernel is shifted three steps forwards. The position of the kernel after the first striding is outlined in orange. When stepping to the third position, the kernel needs to start from the beginning of a new row since there is no room to move further on the same row. The vertical stride of three pixels is now put into practice causing the kernel to be placed according to the purple marking. The final step is taken by shifting the kernel three pixels to the right into the light blue section of the image. A considerably smaller output feature matrix is produced as the result of the convolution because of striding even though a zero-padded image was used. Also, the padded pixels on the right most column and the bottom row

were completely ignored due to their length not being divisible by the size of the kernel.

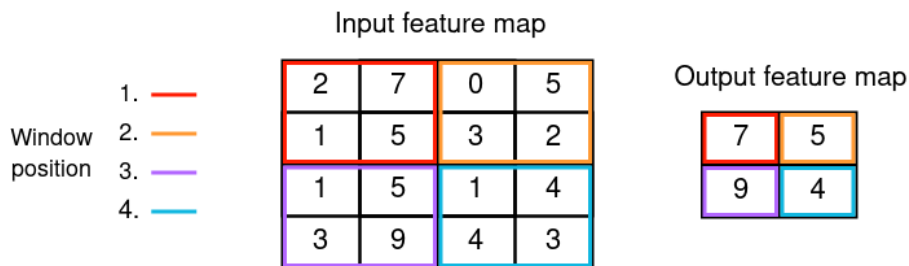
### 2.3.5 Pooling layer

Pooling layers are used as a method to summarise and compress the information in the convolved feature maps. Pooling is a form of non-linear down-sampling that makes CNNs invariant to translation. *Translational invariance* means that regardless of shifting the input or making its appearance vary in some way, the produced response of the network is still the same. This is rather intuitive and a desired property because in image recognition and object detection the specific location of a feature in a feature map is not as interesting as its rough positioning relative to other features. Pooling layers address this issue by progressively reducing the dimensions of the feature maps while preserving the spatial relations of local features. Consequently, all neurons following a pooling layer will have an increased receptive field. Using multiple pooling layers in the feature extraction enables the network to detect more complex data patterns. Another benefit of pooling is the reduced number of network parameters and the amount of computations which allows faster training and makes the network more robust against overfitting while also increasing the generalization performance.

The basic idea of pooling is the following. A pooling kernel is swept over a feature map in the same way as the striding kernel in the previous example from Figure 2.4. In pooling however, instead of applying trainable weights in the kernel, the output is calculated between the batch of pixels within the pooling kernel. After producing an output for a batch of pixels the window is moved forwards. The pooling operation is repeated until the whole feature map has been processed resulting into a compressed version of the input feature map. A stride matching with the size of the pooling window is often used to enable effective down sampling of the input feature map.

There are several different types of pooling operations of which *max pooling* and *average pooling* are arguably the most common. The difference between alternative pooling methods comes down to how the kernel outputs are calculated. In max pooling only a single pixel with the highest value is extracted and stored to the resulting output feature map. In average pooling the average value of the pixels within the kernel is calculated and used as the output. An example of max pooling is shown in Figure 2.5.

In Figure 2.5 the initial position of the pooling window and the placement of the resulting output pixel are marked by the red squares in the corresponding feature maps. For each position of the sliding window the maximum pixel value within the batch of four pixels is extracted to the output feature map. The remaining pixels are discarded. The resulting output feature map is a down sampled version of the



**Figure 2.5** An example of maxpooling with a stride of  $(2,2)$ . The positions of the pooling window at each step are shown in red, orange, purple and light blue rectangles in the corresponding order.

original input feature map. The factor of downsizing is proportional to the size of the pooling window and amount of striding. Having a pooling window with a size of  $2 \times 2$  and a stride of  $(2,2)$  results in an output whose width and height are half the size of the given input.

### 2.3.6 Fully connected layer

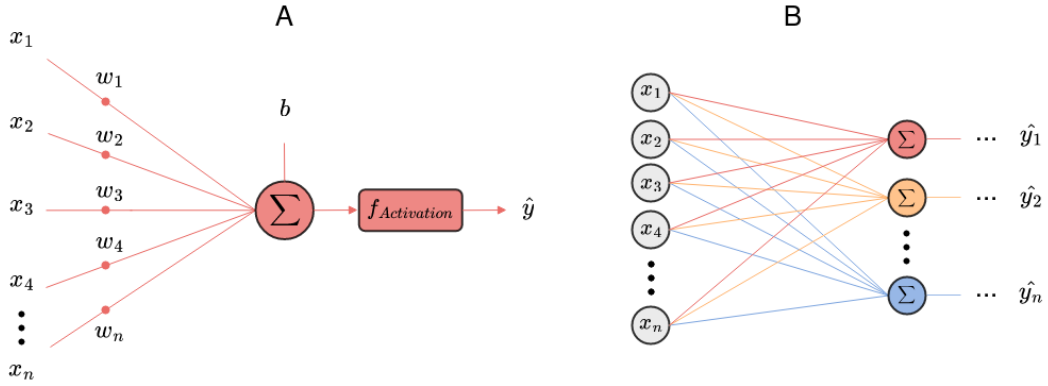
Fully connected layers are familiar building blocks from traditional *Multilayer perceptrons*, but they also have an important role in CNNs. Fully connected layers are used to make the final predictions and decisions based on the information accumulated in the high-level features. This is why fully connected layers are typically found next to the output of a CNN. They are flexible and effective in channeling the extracted feature maps into meaningful number of output nodes while providing a human readable response that is easy to interpret. In other words, fully connected layers are responsible for transforming the high-dimensional data from the high-level features into a meaningful, low-dimensional output.

A neuron, or alternatively a perceptron, is the fundamental building block of fully connected layers. Neurons can be considered as small computing units that are capable of memorizing and detecting different data patterns. The formula of a single perceptron is written as:

$$y = \sigma(w \cdot x + b), \quad (2.9)$$

where  $y$  is the neuron output,  $\sigma$  is a non-linear activation function,  $x$  is a vector of inputs,  $w$  is a vector of input-specific weights,  $\{\cdot\}$  is the dot product operator and  $b$  is a bias term. Neuron's ability to memorize and detect data patterns comes from the trainable parameters  $w$  and  $b$ . The weights can be taught to emphasize or dampen the data in the corresponding input nodes whereas the bias determines the importance of the neuron. Ideally a neuron should produce a high response whenever the input contains some characteristic features regarding the objects of

interests. A visualization of a single perceptron is shown in panel A of Figure 2.6. The behaviour of a neuron is always dependent on the network training which will be discussed in greater detail in chapter 2.5.



**Figure 2.6** An example of a single neuron functionality (A) and a fully connected layer containing several neurons (B).

A fully connected layer, also referred as a dense layer, consists of a set of parallel neurons. The name "fully connected" stems from the fact that each neuron output from a dense layer is connected to all of the neuron inputs at the following layer. An illustration of a fully connected network structure is shown in panel B of Figure 2.6.

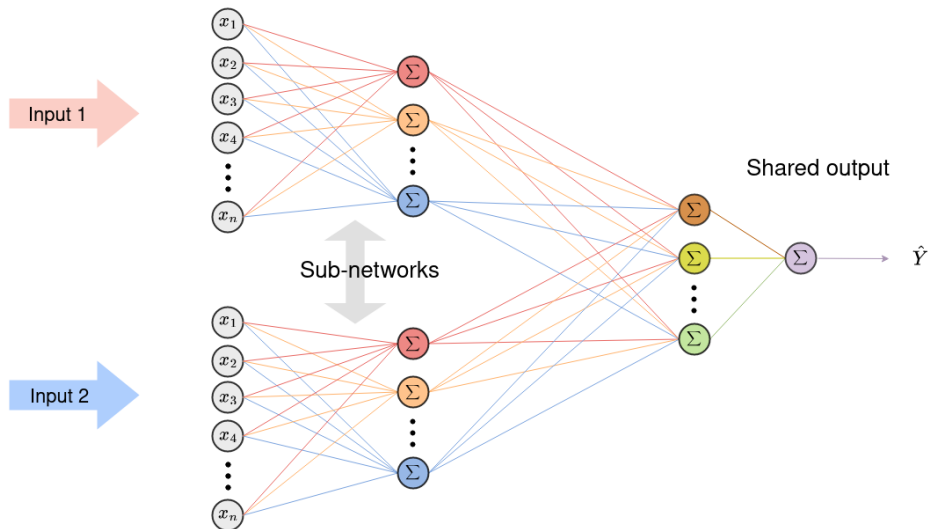
In CNN-based classifiers the final output of the network is often a set of class specific probability values. This can be achieved by using a network architecture with class-specific output nodes  $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N$  and a *sigmoid* activation function that maps each of the produced output values within a closed interval  $[0, 1]$ . The output with the highest probability value can then be chosen as the final networks output  $\hat{Y}$ . The certainty of the prediction can be concluded based on how large the predicted values are. For example, if  $\hat{Y} = 0.9$  for a certain class it means that there is a very strong connection between the input and the said class according to the classifier estimation. Whether or not the produced estimation is to be trusted depends on the training of the model and the results of the evaluation.

## 2.4 Siamese Convolutional Neural Networks

A variety of neural network types can be used to perform *multi-class classification*. However, with most network architectures it is problematic to add new classes for a classifier to be detected once it has been trained. In such cases the whole network would need to be modified or re-trained with an updated dataset each time a new output class is introduced. Another challenging issue is that with some tasks the number of classes may be unknown or beyond comprehension. Song identification

is an example of such scenario where thousands of entirely new compositions are released each day, thus making a multi-class classification approach infeasible.

*Siamese neural networks* (SNNs) are capable of solving tasks that require finding a relationship between comparable objects. SNN is a neural network architecture that contains two or more identical sub-networks at the network input. SNNs process the comparable input items simultaneously in separate sub-network branches that are parallel to each other. The number of input items being compared corresponds to the number of sub-networks, and all the sub-networks are identical to each other in terms of their layer design and trainable parameters. The inputs are then merged together by a connection to a common output branch. The output branch uses the high-level features extracted in the input branches to measure the similarity of the given inputs. SNNs perform well in similarity measurement tasks because there are fewer parameters to learn during the training stage due to the shared weights. SNNs can also produce good results with a relatively small amount of training data. An example of a SNN architecture is shown in Figure 2.7.



**Figure 2.7** The basic structure of a SNN. Neuron biases and activation functions are not shown in the figure for simplicity.

Siamese convolutional neural network (SCNN) is an advanced version of SNN that contains CNNs as part of the network architecture. CNNs are usually applied as the sub-network branches in the model input to enable effective processing of two dimensional input data. Moreover, by making use of the advantages of CNNs it is possible to use a SCNN to measure the similarity of image-like objects. Examples of real-life applications using SCNNs vary from sign language recognition to visual tracking and visual surveillance[25][38][19]. It is also possible to compare audio data and to measure the similarity of the musical content within the different music tracks [16].

## 2.5 Training Siamese Convolutional Neural Networks

As with any deep neural network based system, designing the training process of the model has a key role in the development of the system. Given the task of similarity learning and having large amounts of labelled data available, it is beneficial to apply supervised learning practices in the training. In this section the process of supervised similarity learning is presented with the goal of training a SCNN-based model. The discussed methods are later used in the implementation of the proposed model in chapter 3.

Supervised learning tasks are often divided into three separate stages that are *training*, *validation* and *evaluation*. Given a labelled dataset  $D$ , the data is first split into corresponding subsets  $D_{train}$ ,  $D_{val}$  and  $D_{eval}$ . The idea in training is to gradually develop the model by going through the examples in  $D_{train}$  while making small adjustments to the network to improve its performance. Training and validation are done sequentially in the respective order and each iteration of going through all the data in  $D_{train}$  and  $D_{val}$  is referred as *an epoch*. The purpose of validation is to provide information about the learning progress during the training. The validation examples are not used to update the model parameters at any point to guarantee an unbiased validation. At the end of training the final performance of the model is tested in evaluation against the data in  $D_{eval}$ .

### 2.5.1 Loss Function

The training starts by initializing the network weights and biases with random values. The model is then fed with pairs of input features from  $D_{train}$  along with the desired outputs the model should ideally produce. A pair of input features  $(X_1, X_2)$  contains two separate objects that are either similar or dissimilar to each other. The information about the similarity of the input objects is denoted by a target label  $y \in \{0, 1\}$ . For each input pair the model then computes an output prediction  $\hat{y} \in [0, 1]$  that estimates the similarity of the inputs as a probability value. Each output  $\hat{y}$  is compared to the corresponding target label  $y$  by calculating *the loss* of the output prediction. The loss is a measure of error that indicates how inaccurate a model predictions is. The loss is produced by *a loss-function* or *error function* that is designed to penalize the system for making bad predictions. This means that the greater the difference between the prediction and the expected value is, the greater the produced loss. Examples of commonly used loss functions in similarity learning include *contrastive loss*, *triplet loss* and *lifted structure loss*. However, in situations where the similarity of objects can be expressed as a logical truth value, it is possible to use *binary cross-entropy* (BCE) loss function to derive the loss. The

formula of the BCE-loss for a single prediction is written as:

$$E_{BCE} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})). \quad (2.10)$$

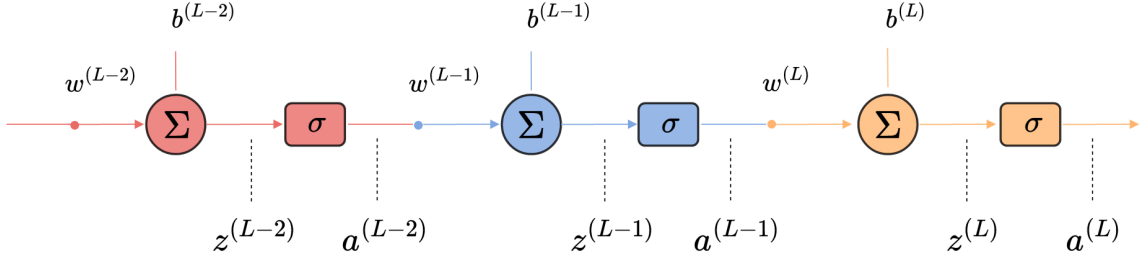
In general, the smaller the losses of predictions are the better the classification performance of the model is. Thus the goal in the training is to minimize the loss over the examples in training and validation sets. The average loss over the whole training set is referred as the *cost function*. Note that loss function and cost function are not the same as loss function considers only the loss of a single observation. So, in order to achieve the goal of minimizing the cost function, the network needs to know how significantly each neuron is affecting to the result. This is where feedforward neural networks make use of the *backpropagation* algorithm.

Before proceeding to the topic of backpropagation, the following explanation about the cost function optimization problem is worth considering. The cost function of a network can be thought as bumpy surface in a coordinate system where each network weight and bias has its own axis. The total loss produced by the training examples on one axis is used to determine the "height" of the surface in this multi-dimensional vector space. Now the random initialization of the network that is done in the beginning of the training will point at a random spot on the surface. The goal from there is to find a combination of weights and biases that leads to the lowest possible spot on the surface and minimize the loss. But in order to do so the network must first find out which direction in this vector space points "downhill". Given that gradients are vectors pointing at the direction of steepest ascent, the opposite of that points in the direction of the most rapid decrease.

## 2.5.2 Backpropagation

Backpropagation is a popular training algorithm that is used in the training of neural networks [29]. In backpropagation the loss produced by the model is used to calculate the gradient of the cost function with respect to all the weights and biases in the network. The partial derivatives computed for the gradient at each layer are reused in the computations at the following layers. Propagating the loss information backwards allows more efficient computation of the gradient compared to a naive approach of computing the gradient of each layer separately. Backpropagation is needed when minimizing the cost function as it produces the gradient vector whose elements show the model whether to increase or decrease the corresponding weights or biases assigned to them.

The math behind backpropagation can be demonstrated by considering a simplified network architecture with only three neurons as shown in Figure 2.8. Given the task of binary classification and using the BCE-loss as the loss function, back-



**Figure 2.8** An example network visualizing the notations used in the backpropagation. The network has only one neuron at each layer.

propagation can be derived as follows. Let  $a$  be the activation of a single neuron and let superscript  $L$  indicate the layer number. Therefore, the activation of the output layer is written as  $a^{(L)}$  while the activation of the previous layer is  $a^{(L-1)}$ . Let  $z$  denote the intermediate output of a neuron before the non-linear activation function has been applied. By following the above-mentioned notations and using Equation 2.9, the output of the network can be written as:

$$a^{(L)} = \sigma(z^{(L)}) = \sigma(w^{(L)}a^{(L-1)} + b^{(L)}). \quad (2.11)$$

The cost function for a single training example can then be obtained by reformulating the BCE-loss in Equation 2.10 such that

$$C_0 = -(y \log(a^{(L)}) + (1 - y) \log(1 - a^{(L)})). \quad (2.12)$$

As stated in Equation 2.11, an activation of a neuron is dependent on the weight  $w^{(L)}$ , the bias  $b^{(L)}$  and the output of the previous neuron  $a^{(L-1)}$ . Given these three factors the first goal is to find out how sensitive the cost function is to changes in each variable. Starting from  $w^{(L)}$ , the derivative of the cost function with respect to the output neuron's weight is  $\frac{\partial C_0}{\partial w^{(L)}}$ . By including the intermediate steps from Equation 2.11 and following the chain rule, the ratio of the derivatives can be written as:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial C_0}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}. \quad (2.13)$$

Similarly, the partial derivatives of the cost function with respect to the bias and the previous layer's neuron can be concluded by replacing  $w^{(L)}$  in the Equation 2.13 with the variable in question. The results of these computations can then be used to determine the respective parameters at the hidden layer. By recursively repeating the same process it is possible to go through the whole network all the way to the input layer. As a result, all of the elements in the gradient vector are obtained for the given training example.

In practice the backpropagation algorithm is used with much more complex net-

works that contain multiple neurons at each layer and where each neuron can have multiple connections to neurons at the adjacent layers. The idea of backpropagating the loss still remains the same and one must only keep track of each individual connection and take them into account when computing the partial derivatives. However, the fact that neurons can influence the output and the cost function through multiple different paths affects the calculation of the previous layer outputs. Instead of having one neuron, the sum of all the neurons at layer  $L$  needs to be computed such that:

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial C_0}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}}, \quad (2.14)$$

where  $j$  indicates the index of the neuron at layer  $L$ ,  $n_L$  is the number of neurons at layer  $L$  and  $k$  denotes the index of the neuron at layer  $L - 1$ . Note that all of the above equations regarding backpropagation have focused on determining the gradient of the cost function based on a single training example. In order to get the true gradient of the cost function every input-output example in the training dataset needs to be considered. For each weight and bias in the network the mean of the partial derivatives over the whole training dataset are used to form the final gradient vector:

$$\nabla C = \left[ \frac{\partial C}{\partial w_1^{(1)}} \quad \frac{\partial C}{\partial b_1^{(1)}} \quad \cdots \quad \frac{\partial C}{\partial w_{n_L}^{(L)}} \quad \frac{\partial C}{\partial b_{n_L}^{(L)}} \right]^T. \quad (2.15)$$

### 2.5.3 Gradient descent

*Gradient descent* (GD) optimization algorithm is used together with backpropagation to update the model weights and biases during the training. The goal in GD is simply to navigate to the minimum point of the cost function. This is done gradually by iterating over the training data and repeatedly computing the negative gradient of the cost function and adjusting the model parameters accordingly. There are three alternative approaches for applying GD that are *stochastic gradient descent* (SGD), *batch gradient descent* and *mini-batch gradient descent*. In SGD the computations of gradients are done most frequently as the losses of each training example are used individually to update the model weights. In batch GD the mean loss of the entire training set is used for computing the gradient rather than updating the weights after each forward pass. In mini-batch GD the set of training examples are divided into smaller batches. The gradients are computed based on the mean loss of each batch, so the number of weight optimizations during an epoch is equal to the number of batches.

While backpropagation provides the direction in which the weights and biases need to be adjusted, there is also the question of how big of a change or step along that direction is sufficient. For this, a *learning rate* coefficient  $\alpha$  is introduced to

describe the step size that is taken every time the model parameters are updated. Let  $\theta_t$  denote the configuration of the trainable network parameters at iteration  $t$ . The GD algorithm can then be formulated as

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial C(D_{train}, \theta_t)}{\partial \theta}. \quad (2.16)$$

Having a too small learning rate  $\alpha$  can make the learning progress impractically slow. On the other hand, taking too large steps can make it impossible to find a local minima of the cost function. Several extensions and variants have been developed to balance between inefficient time consumption and the potential risk of constantly overstepping the optimal values. *SGD with Momentum*, *Adam* and *AMSGrad* are examples of other optimizers that include an adaptive learning rate mechanism [33][17][27]. Adaptive learning rate is able to take bigger steps whenever the gradient is steep, thus speeding up the training. Conversely, smaller and more careful steps are taken when the gradient approaches to zero. The computational details of more advanced optimizers are not discussed in this thesis.

## 2.6 Frequency Domain Presentation of a Signal

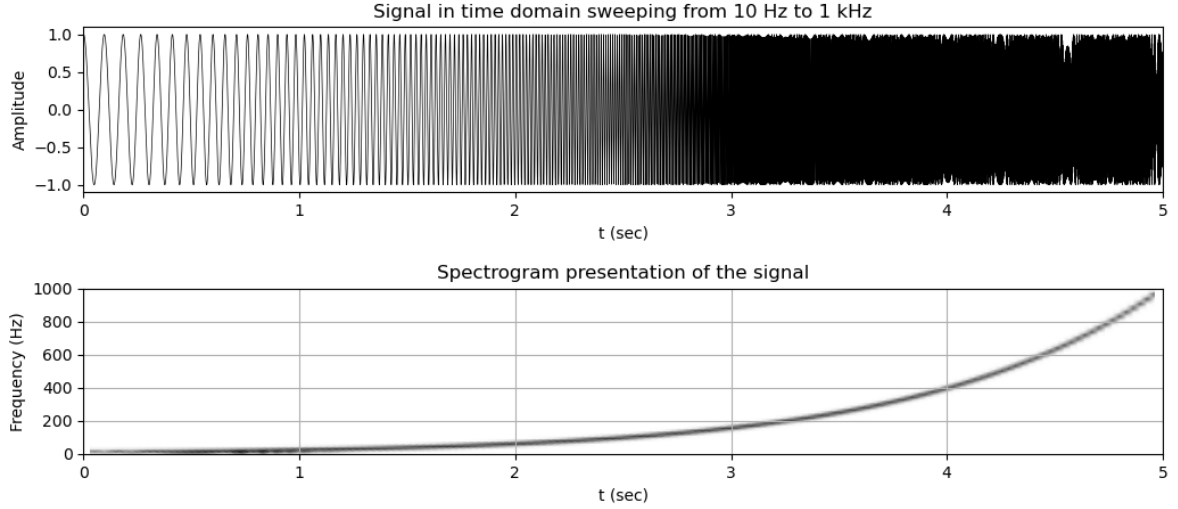
A digital recording of a sound signal is practically a collection of voltage values from a microphone that are converted and stored into a digital form at a regular sampling frequency. Plotting the measured amplitudes of a signal as a function of time is generally known as the *time domain presentation*. Signals can be also presented in *frequency domain* to show what frequencies are present in an examined signal. Decomposing a signal to its frequency components can be achieved for example by using Fourier related transforms such as the *short-time Fourier transform* (STFT). Frequency domain presentations in general are much more useful and informative when it comes to spectral analysis or modifying the contents of a signal. Information about the phase shifts along the spectrum can be also examined better in frequency domain.

This section presents two of the many existing frequency transforms that are the STFT and constant Q-transform (CQT). The basics of the STFT are first discussed in Section 2.6.1. An introduction to CQT is then provided in Section 2.6.2.

### 2.6.1 Short-time Fourier Transform

The STFT has multiple use cases in digital signal processing of which the *spectrogram* presentation of a signal is probably the most evident example. A spectrogram of a signal visualizes the temporal evolution of different frequencies as a function of time. It provides time-localized information about the overall energy distribution

across the measured spectrum in an easily interpretable graph. An example of a spectrogram is shown in Figure 2.9. A spectrogram is obtained by calculating the squared magnitude of the STFT of a signal.



**Figure 2.9** Example of a spectrogram presentation of a signal. The spectrogram in the lower panel shows the logarithmic sweeping of the time-domain signal

In the STFT a signal in time domain is first divided into short overlapping segments. Each short segment is then filtered by a windowing function to force the samples near the segment boundaries to be close to zero. Windowing and overlapping are used to minimize spectral leakage in the frequency domain and to improve the overall accuracy of the presented spectrum. In the STFT the time-frequency transform is done by applying *discrete Fourier transform* (DFT) to each filtered segment. DFT aims to find a set of sinusoids that can be added together to produce the examined time domain segment. This includes determining the amplitude and phase coefficients for each frequency component. In practice the DFT is calculated by using the *fast Fourier transform* algorithm. It reduces the complexity of the transform and speeds up the process, which makes it possible to use the transformation in real-time applications.

The STFT of a signal is a complex valued frequency domain representation and a function of two variables, time and frequency. The STFT of an input signal  $x[n]$  can be described as:

$$STFT\{x[n]\}(m, \omega) \equiv X[m, \omega] = \sum_{n=-\infty}^{\infty} x[n]w[m-n]e^{-j\omega n}, \quad (2.17)$$

where  $m$  is the frame index,  $w[m-n]$  is the windowing function and  $\omega$  is the continuous frequency variable. The notation  $X[m, \omega]$  is used to highlight the fact that  $m$  is discrete whereas  $\omega$  is continuous. The numerical evaluation of the STFT

requires sampling the frequency axis  $\omega$  in  $N$  equally spaced samples from  $\omega = 0$  to  $\omega = 2\pi$ . The division of  $\omega$  into  $N$  discrete frequency channels is therefore written as:

$$\omega_k = \frac{2\pi}{N}k, \quad \forall k : 0 \leq k \leq N - 1. \quad (2.18)$$

Let  $N_w$  specify the length of the finite window function  $w[m - n]$ . By combining the equations 2.17 and 2.18, the discrete STFT can now be written as:

$$X[n, k] = \sum_{n=m-(N_w-1)}^m w[m - n]x[n]e^{-j\omega_k n} = \sum_{n=m-(N_w-1)}^m w[m - n]x[n]e^{-\frac{j2\pi nk}{N}}, \quad (2.19)$$

where  $k$  is the frequency bin index and the location of the analysis window along the time axis is denoted by  $m$ .

There is a trade-off between the frequency resolution and the time resolution when computing the STFT. Narrow analysis windows provide good time resolution but poor frequency resolution, whereas wider windows give better frequency resolution but poor time resolution. The frequency resolution can be calculated as the sampling rate divided by the size of the windowing segments in samples. At any case the uniform sampling of the frequency axis  $\omega$  in  $N$  frequency bins causes the frequency resolution to be a constant across the whole spectrum. This property is not ideal for musical applications nor for emulating human perception because human auditory system corresponds more closely to a logarithmic frequency scale rather than linear [28]. Most importantly, the problem with the STFT is that the frequency resolution is not accurate enough at low frequencies, while at the same time the frequency resolution at the highest bins is too accurate for the purpose of detecting harmonic frequency components of musical notes.

The inefficiency of Fourier related transforms in musical note detection can be demonstrated by considering the following example. In the western music scale a semitone is the smallest step from one note to another and each note has a unique fundamental frequency. Therefore, one should be able to determine what musical note an instrument is playing based on a spectral analysis of such recording. Given that sounds consisting of harmonic frequency components will form a constant pattern in the frequency domain when plotted against log frequency and that the pattern remains the same independent of the fundamental frequency, musical note detection becomes a simple pattern recognition task in the frequency domain.

In Table 2.1 the fundamental frequencies of adjacent musical notes 'E' and 'F' are presented at eight different octaves. The differences of the two notes at each octave are also shown in the bottom row. When looking at the difference in the first octave, the fundamental frequencies of adjacent notes  $E_1$  and  $F_1$  differ from each other only by 2,45 Hz. Then again, when looking at the corresponding values

| Octave               | 1     | 2     | 3      | 4      | 5      | 6       | 7       | 8       |
|----------------------|-------|-------|--------|--------|--------|---------|---------|---------|
| $f_0^E$ (Hz)         | 41,20 | 82,41 | 164,81 | 329,63 | 659,25 | 1318,51 | 2637,02 | 5274,04 |
| $f_0^F$ (Hz)         | 43,65 | 87,31 | 174,61 | 349,23 | 698,46 | 1396,91 | 2793,83 | 5587,65 |
| $f_0^F - f_0^E$ (Hz) | 2,45  | 4,90  | 9,80   | 19,60  | 39,21  | 78,40   | 156,81  | 313,61  |

**Table 2.1** The fundamental frequencies of musical notes 'E' and 'F' at different octaves.

in the eighth octave, the same step of one semitone from  $E_8$  to  $F_8$  has 313,61 Hz difference between their fundamental frequencies. Generally speaking, the difference between two adjacent musical notes in lowest octaves is only a matter of hertzes, whereas in higher octaves the corresponding difference in frequency is about two orders of magnitudes greater. Having a constant frequency resolution is therefore either inaccurate or ineffective given the logarithmic nature of musical notes.

### 2.6.2 Constant Q-Transform

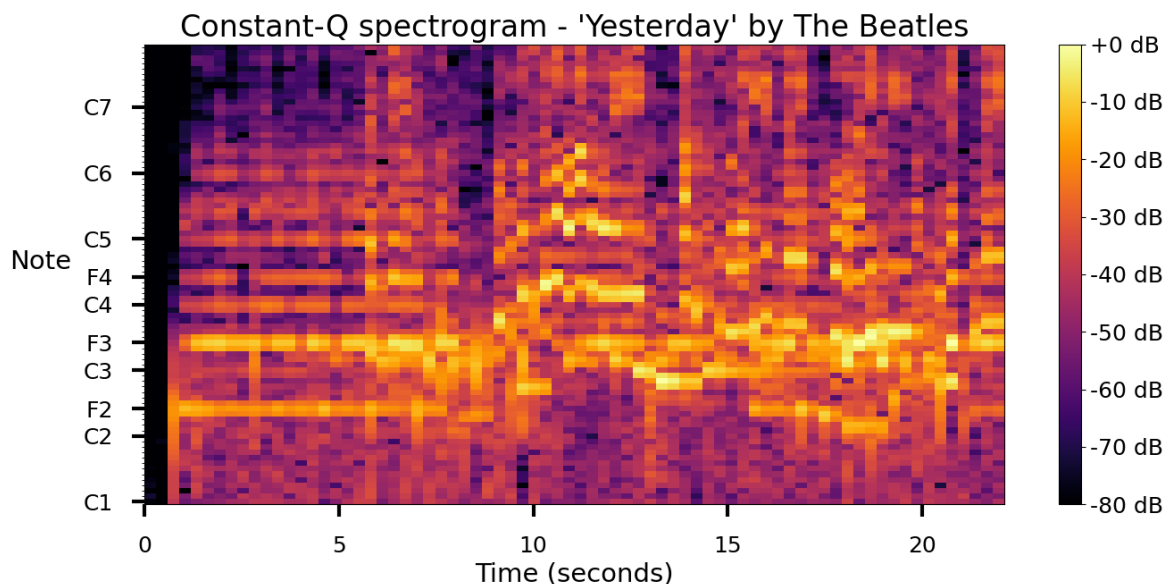
The CQT is a type of frequency domain transformation that has been specifically designed for musical applications [4]. It is suitable for analysing harmonic signals because it solves the problem of constant frequency resolution in Fourier related transforms. The main difference between the CQT and the STFT is that in the CQT the size of the analysis window is altering according to the examined frequency. Larger window sizes are used for lower frequencies while smaller sized windows are used for higher frequencies. The name 'constant Q' comes from the constant ratio of frequency  $f$  to frequency resolution  $f_\delta$  that is simply given as:

$$Q = f/f_\delta. \quad (2.20)$$

The CQT provides better results compared to the STFT when low frequencies and logarithmic frequency mapping are concerned. CQT also enables more effective distinction of adjacent notes in a musical scale at higher frequencies, thus making it a suitable feature presentation method for detecting melodic patterns and chord progressions. However, the CQT is computationally more expensive which is not optimal for real-time applications. It has also slightly less detail in the far upper frequencies and it is not invertible like the STFT. These drawbacks are acceptable given the task of song identification from live music data in an offline environment.

Applying the CQT to a piece of music and plotting the *constant Q spectrogram* (CQ-spectrogram) enables musical analysis of the song based on visualized data. For example, it is possible to detect melodic patterns from the CQ-spectrogram irrespective of the musical key, tempo or instrumentation used in the song. By comparing two CQ-spectrograms of different tracks it is possible to determine if the different tracks contain similar structural or melodic patterns. Furthermore, finding

similar melodic patterns can be used to conclude if both of the tracks are different versions of the same song. An example of a CQ-spectrogram of a song track is shown in figure 2.10.



**Figure 2.10** An example of a constant Q spectrogram extracted from the track 'Yesterday' by The Beatles. In the CQT the number of frequency bins per octave is 12, the minimum frequency is 32.7 Hz and the window length as well as the hop size is 6592 samples.

When analyzing the CQ-spectrogram in figure 2.10 it has several recognizable features that match with the perceived audio content in the song track. For example, as in many mp3-files there is the so called 'mute time' in the beginning of the track that is seen as silent columns of data at around 0 seconds. The instrumental intro of the song has an acoustic guitar playing the notes  $F_2$ ,  $F_3$ ,  $C_4$  and  $F_4$ . The corresponding note frequencies are thus seen in the CQ-spectrogram as orange horizontal lines that show the presence of the notes and their intensity. The repetitive strumming of the guitar strings and the long sustain of the played notes explain the stationary patterns during the first five seconds of the song. The harmonic components of the notes in the intro are also visible in the fifth and sixth octave. The most interesting feature in terms of automatic song recognition is the song defining vocal melody pattern between the notes  $F_3$  and  $F_4$  at around 9-13 seconds. Also the harmonic components of the notes forming the melody are seen one octave above.

## 2.7 Cross-similarity Matrix

A matrix as a mathematical object is a table of elements consisting of rows and columns. The elements or entries of a matrix can be numbers, symbols or expres-

sions. Many of the properties in linear algebra have been defined for matrices which makes them a powerful tool for expressing and solving multi-variable problems. As an in-depth introduction to matrix calculus is beyond the scope of this thesis, the following paragraphs are meant to briefly introduce and derive the concept of *cross similarity matrix* (CSM). Being familiar with the basics of CSMs is needed to fully understand the operating principles of the proposed system.

*Similarity matrix* is a term used about matrices that quantify the similarity between two objects. The elements of a similarity matrix can be considered to be score values that express the similarity between two individual data points. The way in which similarity matrices are calculated can vary as there are no strict rules defining the process. Generally speaking, a similarity matrix can be viewed as the inverse of a distance matrix, meaning that large values are produced for similar objects while lower or negative scores indicate dissimilarity. By analysing the elements of a similarity matrix one can make predictions about local or global structural similarities between the examined objects. Similarity matrices are commonly used in sequence alignment which has use cases in the field of bioinformatics and music information retrieval.

CSM is a specific implementation of a similarity matrix. CSMs have been recently used in the task of CSI and it is derived as follows. Let vectors  $A = \{a_1, a_2, \dots, a_N\}$  and  $B = \{b_1, b_2, \dots, b_N\}$  present the examined input data sequences where  $N$  is the total number of elements in each vector. Let  $C_{(i,j)} \in \mathbb{R}^{N \times N}$  denote the CSM where  $i, j \in \{1, \dots, N\}$  are row and column identifiers as well as the element indexes of vectors  $A$  and  $B$ , respectively. The equation for calculating the CSM of vectors  $A$  and  $B$  can then be expressed as:

$$C_{(i,j)} = \|a_i - b_j\|^2. \quad (2.21)$$

### 3 Live Music Song Identification System

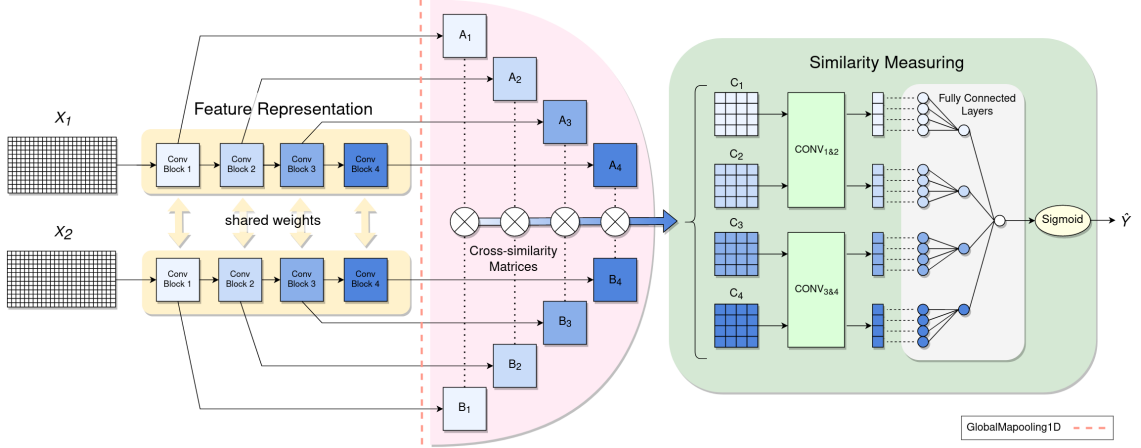
The third chapter presents the proposed model architecture used in the task of live music song identification. The chapter goes through the different components of the model following the flow of data starting from the input and proceeding towards the output. General overview about the model is first provided in Section 3.1. Information about the input features, feature extraction and feature representation are then presented in Sections 3.2 and 3.3. The way in which CSMs are used in the model is explained in Section 3.4 followed by the details of similarity measurements in Section 3.5.

#### 3.1 System Overview

The system consists of two components that are a music database of unique musical compositions and a SCNN-based model. Given an audio track of a live recording of a song, the system aims to retrieve the corresponding studio version of the song from the music database. Live recordings are entered to the system as input queries and for each query the model is used to measure the musical similarity between the query item and the songs in the database. The model takes a pair of audio tracks as its input at a time, one being the query item and the other a song from the database. The model outputs a similarity score that quantifies the musical similarity of the input tracks as a probability value. After computing the similarity score for each song in the database, a relevance-based ranking of the songs is formed. The studio version with the highest similarity score is considered as the most relevant item and retrieved by the system as the final output.

The proposed model architecture is shown in Fig. 3.1. The model is a cascade of three jointly trained networks that makes use of CSMs of *multi-level deep sequences*. Multi-level deep sequences in the context of SCNNs mean that there are multiple output connections at varying layer depths along the identical input branches of the SCNN. The main benefit of this method is that no handcrafted features are needed, meaning that no information is lost during the feature extraction stage [16]. The method is also robust against changes in tempo and key transposition, which is a crucial property when dealing with live music data.

The learning setup is done by training the model with pairs of similar and non-similar objects as described in Section 2.5. Now the similar objects are pairs of audio tracks containing the same underlying song with alternative presentations, while non-similar objects are pairs of tracks presenting different musical pieces. The labelling of similar and non-similar examples is done such that similar objects are



**Figure 3.1** A block diagram of the model architecture. In model input CQ-spectrograms  $X_1$  and  $X_2$  from two separate tracks are fed to different branches of the SCNN performing the feature representation. Resulting multi-level deep sequences  $A_1 \dots A_4$  and  $B_1 \dots B_4$  are then used to compute level-specific CSMs  $C_1 \dots C_4$ . Two parallel CNNs are used for similarity measuring, followed by four fully connected layers, a single output neuron and a sigmoid activation function. The final output is a similarity score  $\hat{Y} \in [0, 1]$  that describes the similarity of the given input tracks as a probability value.

assigned as class 1 and non-similar objects are referred as class 0. The training dataset  $D_{train}$  consist of pairs of features of songs  $(X_1, X_2)$  and ground truth labels  $Y \in \{1, 0\}$  that signify whether or not the paired tracks present the same underlying song, respectively. BCE-loss is used in the training as the loss-function. Let  $\theta$ ,  $\gamma_1$  and  $\gamma_2$  denote the parametrization of the SCNN and the parallel CNNs in the similarity measuring accordingly. The objective in the training can therefore be written as

$$\arg \min_{\theta, \gamma_1, \gamma_2} \sum_n -Y_n \log(\hat{Y}_n) - (1 - Y_n) \log(1 - \hat{Y}_n), \quad (3.1)$$

where  $\hat{Y} \in [0, 1]$  is a predicted similarity score produced as the model output and  $n$  denotes the index of the paired song examples in the dataset  $D_{train}$ .

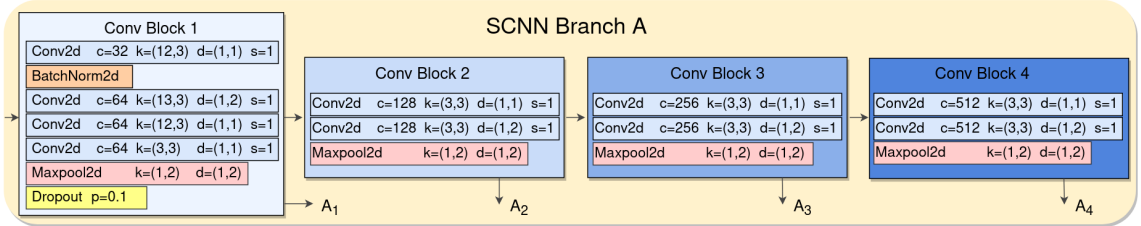
### 3.2 Feature Extraction

In feature extraction, a fixed duration of 120 seconds of audio is extracted from the beginning of each track. Songs with a duration less than two minutes are zero padded to have the same length. The audio tracks are downsampled to have a sampling rate of 22050 Hz and converted from stereo to mono signals by averaging the samples across the stereo channels. CQ-spectrograms are then computed from the time-domain audio signals by taking the absolute values of CQTs. For each CQ-spectrogram  $X \in \mathbb{R}^{N \times T}$  the number of frequency bins  $N = 84$  and the number of time-frames  $T = 401$ . In the CQT the number of frequency bins per octave is 12,

the minimum frequency is 32.7 Hz and the hop length is 299 ms. The features of each frequency bin and track are standardized separately to have a mean of 0 and a standard deviation of 1.

### 3.3 Feature Representation

The SCNN in the model input is comprised of two identical CNN-branches  $A$  and  $B$  that share the same weights and parameters. At first, the input features  $(X_1, X_2)$  are fed to the separate branches of the SCNN. The branches consist of four blocks of convolution and maxpooling operations as shown in Fig. 3.2. Depending on the branch, each convolution block produces a multi-level deep sequence  $A_k = \{a_1^k, a_2^k, \dots, T_k^k\}$  or  $B_k = \{b_1^k, b_2^k, \dots, T_k^k\}$ , where  $k \in \{1, 2, 3, 4\}$  denotes the level of deep sequence,  $T_k$  is the width of a convolved CQ-spectrogram at  $k$  and  $A_k, B_k \in \mathbb{R}^{N_k \times T_k}$ , where  $N_k$  is the number of convolution channels at  $k$ . Initially the convolution blocks produce three-dimensional tensors having a shape of  $N_k \times H_k \times T_k$ , where  $H_k$  is the height of the convolved CQ-spectrograms. At the end of the feature representation, a Global Maxpooling1d-operation is applied to each block output along the frequency axis, thus collapsing  $H_k$  to 1 and reducing the dimensions of  $A_k$  and  $B_k$  to  $N_k \times T_k$ .



**Figure 3.2** The inner structure of branch A of the SCNN. The data flow and the parametrization of the four convolution blocks are identical in branch B. The layer parameter names are abbreviated as follows:  $c$ =number of channels,  $k$ =kernel size,  $d$ =dilation,  $s$ =stride and  $p$ =dropout probability.

The use of deep sequences enables examining and comparing the input songs at different abstraction levels. Multiple levels of granularity presented by the deep sequences provide more information for measuring the similarity of the songs compared to a traditional structure of a SCNN-branch with a single output. More details about the structure of the SCNN are shown in Fig. 3.2.

### 3.4 Cross Similarity Matrices

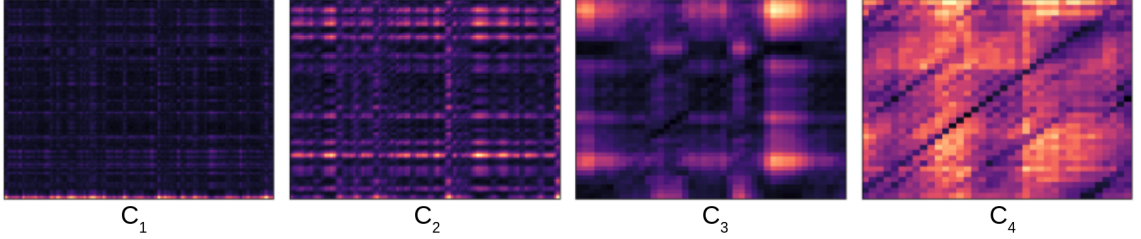
The parallel branches of the SCNN are connected by computing CSMs from the extracted multi-level deep sequences  $A_k$  and  $B_k$ . Level-specific CSMs  $C_k \in \mathbb{R}^{T_k \times T_k}$  are calculated for each  $k$  as

$$C_{k,(i,j)} = \|a_i^k - b_j^k\|^2, \quad (3.2)$$

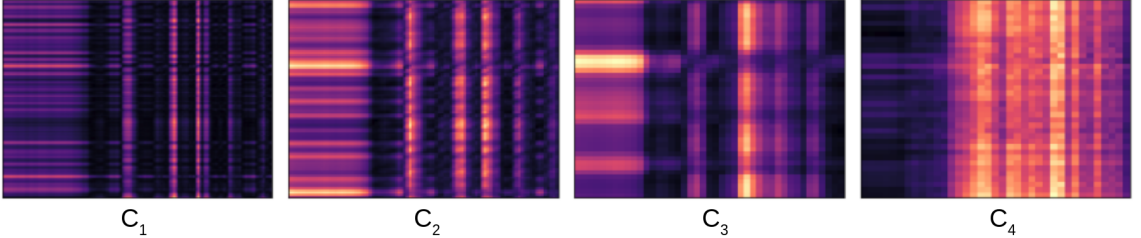
where  $i, j \in \{1, \dots, T_k\}$ .

The resulting CSMs can be viewed as rectangular images consisting of  $T_k^2$  pixel values. Examples of such image presentations are shown in Fig. 3.3. Any two input tracks with similar musical content will produce diagonal patterns to the CSMs as demonstrated by the first row of images in Fig. 3.3. Conversely, input songs with dissimilar content result in grid-like shapes. The patterns formed on CSMs will ultimately determine the similarity or dissimilarity of the two input tracks.

Musical similarity: Input songs of class 1



Musical dissimilarity: Input songs of class 0

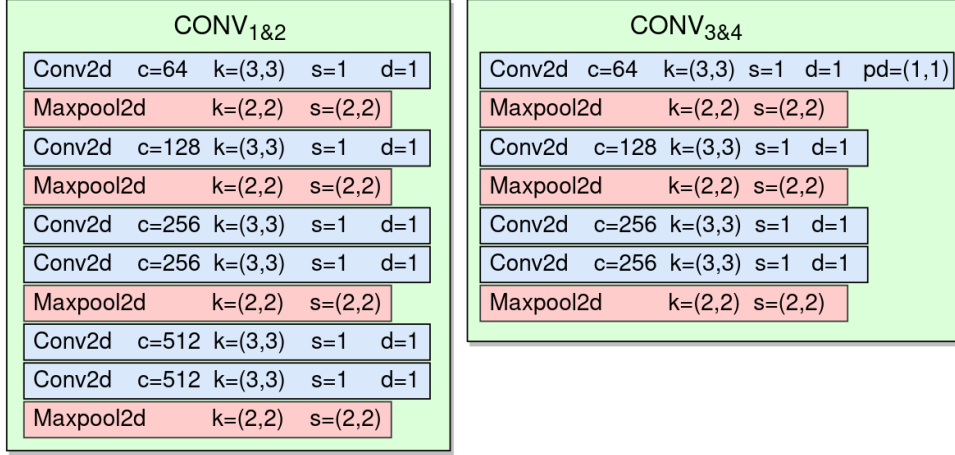


**Figure 3.3** Visualization of CSMs. The input songs used in class 1 example are ‘Losing My Religion’ by R.E.M and a live recording of the same song. In class 0 example the same live recording track is compared against ‘E5150’ by Black Sabbath.

### 3.5 Similarity Measuring

The CSMs are fed to the similarity measuring stage of the model to examine the musical similarity of the input tracks. There are two parallel CNNs with individual weighing and parametrization to deal with CSMs from different levels of abstraction. The matrices  $C_1$  and  $C_2$  are processed in  $\text{Conv}_{1\&2}$  while  $C_3$  and  $C_4$  are processed in  $\text{Conv}_{3\&4}$ . The details about the inner structures of  $\text{Conv}_{1\&2}$  and  $\text{Conv}_{3\&4}$  are provided in Fig. 3.4.

The above-mentioned approach of treating CSMs from different levels of abstraction separately was chosen rather than using one shared CNN for each  $C_k$  as was done in method [16]. Given the significant differences between the sizes of  $C_1 \in \mathbb{R}_+^{194 \times 194}$ ,  $C_2 \in \mathbb{R}_+^{93 \times 93}$ ,  $C_3 \in \mathbb{R}_+^{43 \times 43}$  and  $C_4 \in \mathbb{R}_+^{37 \times 37}$  it seems counter-intuitive to treat the CSMs without taking the matrix resolution into account. Also the fact that  $C_3$  and  $C_4$  tend to contain much more diagonal patterns for inputs of class 1 suggests that it is reasonable to train two CNNs that can focus on level-specific data patterns.



**Figure 3.4** CNN structure and layer parameters. The layer parameter names are abbreviated as follows:  $c$ =number of channels,  $k$ =kernel size,  $d$ =dilation,  $s$ =stride and  $pd$ =padding.

The CNNs are followed by four branches of fully connected neural network layers, each dedicated for processing CSMs of a certain level. The number of neurons at each branch are 32768, 2048, 1024 and 1024 which is the number of pixel values remaining in the convolved and shrunk  $C_k$  accordingly. There are four hidden neurons following the first layer, each connecting the neurons at the same branch to enable different weighing of CSMs. The hidden neurons are then connected to a single output neuron. Finally, a sigmoid function is applied to the output neuron to produce the similarity score  $\hat{Y}$  as the model output.

## 4 Experiments

The results of the experiments are provided in this chapter along with the information about the data and the model hyperparameters. The data used in the model training, validation and evaluation are first discussed in Section 4.1. The details about the hyperparameters of the model are then presented in Section 4.2. The final results of the experiments are provided in Section 4.3.

### 4.1 Data

Data from *Second Hand Songs 100K*-dataset (SHS100K) was used in the training of the model [23]. SHS100K is a large dataset of songs and their freely accessible recordings of cover versions. The dataset has been created for academic use and it has become the benchmark dataset for CSI. SHS100K contains about 10k songs with 100k audio tracks, out of which 6.2k songs and 67.8k tracks were applicable on Youtube. Songs whose original version was not available were excluded from the experiments as well as the songs without any available cover versions.

A diverse range of music genres are included in the SHS100K dataset. For example, a piece of music originally written as a pop rock song can have cover versions in which the genre of the song has been changed to blues, metal, punk, country and jazz. Other common genre examples include stripped down acoustic versions, instrumentals and a cappella covers. The musical differences between alternative versions of the same song can therefore vary significantly. On some occasions the cover versions are very similar to the original song, while sometimes the musical similarities between a cover version and the original song are so vague that the only recognizable thing in common are the lyrics of the song. On average there are 9.9 cover versions per each song in the SHS100K-dataset. The highest number of cover versions per song is 195 and there are 484 songs with only one cover version.

Pre-processing of the SHS100K data was done as follows. At first, examples of class 1 were created by linking each cover version with their corresponding original version. The linked song pairs were then split into training and validation sets with a division of 75%-25%, respectively. Songs with multiple cover versions were put in the same set so that they were only present in either training or validation but not in both. Secondly, examples of class 0 were generated within each set by randomly swapping cover songs between the linked song pairs. In other words all original song versions were coupled with an alternative version of a different song. A class distribution ratio of 1:3 was used between the class 1 and class 0, meaning that the number of dissimilar objects was three times higher than the number of song pairs

presenting the same song.

|              | SHS100K | Covers80 | Live Music Dataset |
|--------------|---------|----------|--------------------|
| Audio tracks | 61,6k   | 160      | 678                |
| Song pairs   | 55,4k   | 80       | 407                |
| Artists      | 31,3k   | 92       | 37                 |
| Live music   | No*     | No       | Yes                |
| Audio format | mp3     | mp3      | wav                |
| Raw audio    | 340 GB  | 152 MB   | 28 GB              |
| Features     | 8 GB    | 20 MB    | 91 MB              |

**Table 4.1** *Statistics and information about the used datasets.*

A custom live music audio dataset was used for the model evaluation. The dataset was collected manually from the cd-album collections of PIKI online library [24]. The goal for the data collection was to gather as many studio recording-live recording song pairs as possible. The Live Music Dataset contains dozens of artists and a variety of music genres from several decades. In majority of collected studio version-live version pairs the performing artist is the same in both of the tracks. The full list of songs collected is presented in the Appendix in Table Appendix.1. Covers80 dataset was also used in the evaluation of the model [9]. Covers80 consists of 80 songs, each performed by two artists. The use of the dataset provides a common ground for comparing the model performance against previous CSI-methods in the literature. All in all, the songs and their cover version in the Covers80-dataset are quite similar with the songs in the SHS100K-dataset in terms of musical changes. One major difference however is that the data format of the songs in the Covers80-dataset is mp3, whereas the songs in the Live Music Dataset are stored in wav-format. Details about the three datasets used in the model development and evaluation are shown in Table 4.1.

## 4.2 Model Parametrization

The model development was done in Python by using PyTorch machine learning framework. Mini-batch training with a batch size of 8 was used in the training. AMSGrad optimizer algorithm was used with an initial learning rate of 0.001 [27]. The learning rate was decreased gradually during the training by using a learning scheduler with a factor of 0.1 and a patience of 10. This means that if the validation loss does not reach a new minimum value within the number of epochs set by the patience-parameter, the learning rate is reduced according to the learning rate factor. The maximum number of training epochs was set to 150. A holdout validation set

$D_{val}$  was used in the training in order to compute and store the validation losses at the end of each training epoch. The epoch with the minimum validation loss was considered as the optimal state for the trainable parameters. The final values for each hyperparameter were obtained by trial and error. A summarization of the hyperparameters and their values are shown in table 4.2

| Hyperparameter       | Value     |
|----------------------|-----------|
| Train-val split      | 75% - 25% |
| Max epochs           | 150       |
| Learning rate        | 0,001     |
| Batch size           | 8         |
| Shuffle              | True      |
| Sampling frequency   | 22050     |
| Mono audio           | True      |
| Audio duration       | 120s      |
| CQ-spec. time frames | 401       |
| Hop length           | 6592      |
| $F_{min}$            | 32,7      |
| Freq. bins           | 84        |
| Feature standard.    | True      |

**Table 4.2** *The Hyperparameters used in the training.*

### 4.3 Experiment Results

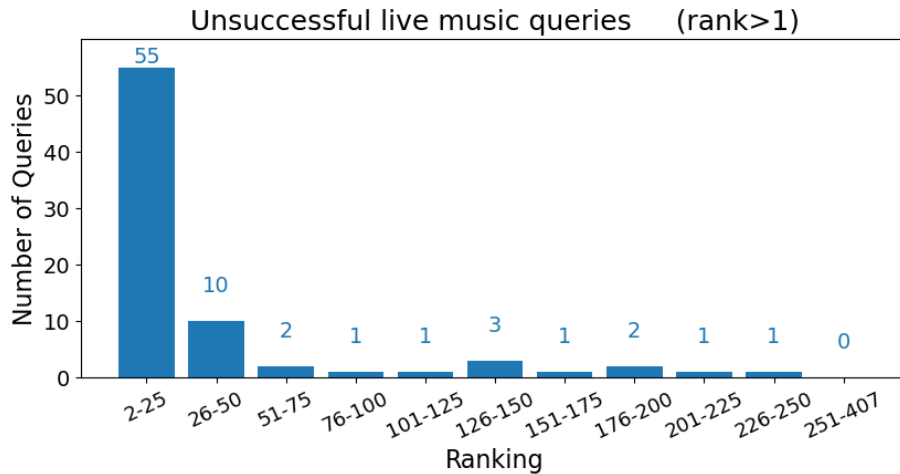
The final results of the experiments are shown in Table 4.3. The evaluation was done separately for two different datasets, the Live music dataset and Covers80. In both experiments the number of relevant items to be retrieved for each query is one. Therefore, the best possible values for  $P@10$  and  $MAP$  are 0.100 and 1.000, respectively. All live versions in the live music dataset were used as the query items. For Covers80-dataset the two song lists to which the songs have been divided were used such that the versions in "list1" were treated as query items while the songs in "list2" were used as the retrievable data.

| Dataset            | P@10  | MR1   | MAP   |
|--------------------|-------|-------|-------|
| Live Music Dataset | 0.092 | 6.838 | 0.846 |
| Covers80           | 0.072 | 9.38  | 0.558 |

**Table 4.3** *The results of the experiments.*

## 5 Discussion

The trained model is capable of identifying songs from live music tracks according to the results of the experiments shown in Table 4.3. The overall performance of the implemented system is rather good considering that the model was trained mostly with non-live music data. Further analysis of the results shows that 81.1% of the queries made with the Live Music Dataset retrieved the desired studio version as the most relevant result ( $r_1 = 1$ ). Moreover, 90.2% of the live music queries retrieve the relevant song among the five most matching predictions. With the Covers80 dataset the corresponding ratios are 50.0% and 61.3% which are unexpectedly less than with live music data. However, after listening to the songs in each dataset one can easily notice that the musical similarities between the class 1 examples in Live Music Dataset are much more obvious than those in the Covers80 songs. The live music songs are almost always performed by the same artists that have recorded the corresponding studio versions and hence the music genre is often the same between the songs to be matched. This is not the case in Covers80 dataset where the performing artists as well as the music genres are always different. The number of artists in each dataset also supports this observation. It is also worth noting that with Covers80-dataset there is only one query per each song while Live Music Dataset has several instances where multiple queries are made with different live recordings of the same song. The details about the number of live versions per each song in the Live Music Dataset is shown in Appendix in Table Appendix.1.



**Figure 5.1** Histogram of rankings of unsuccessful live music queries (rank > 1). The total number of queries is 407, of which 77 (18.9%) are presented.

A histogram presentation about all of the unsuccessful live music queries is shown in Fig. 5.1. The histogram shows the distribution of the ranking results of the

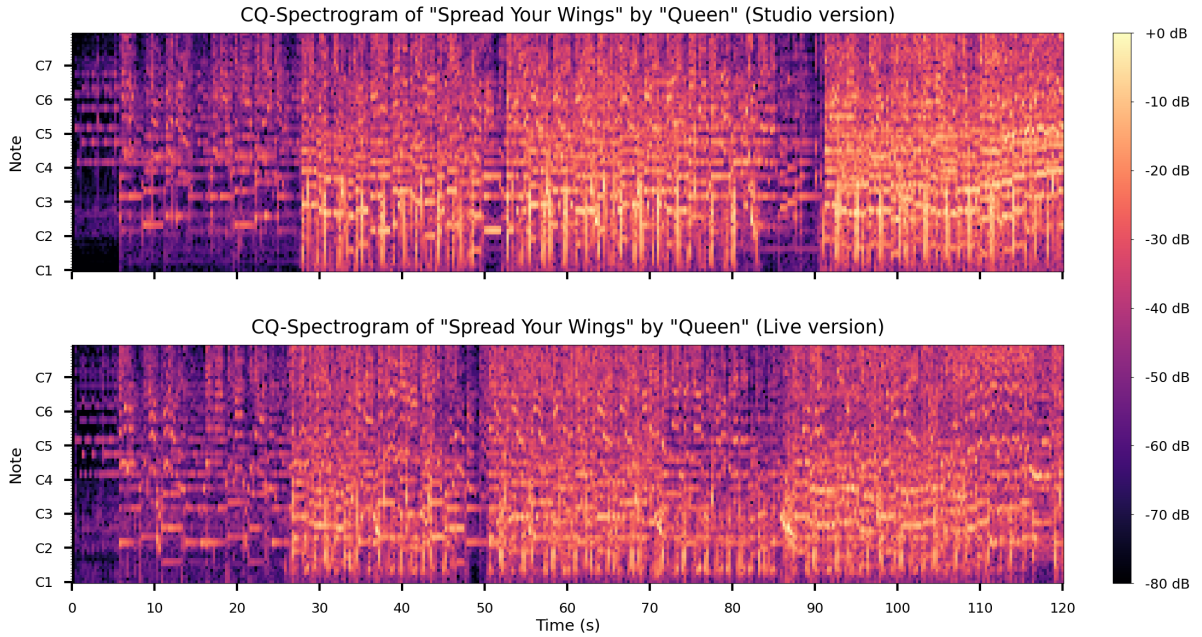
studio versions that the system failed to retrieve as the best match in the music database. With most of the failed queries the similarity score of the relevant item is ranked among the top-25 retrievals. At the same time some queries perform very poorly in measuring the song similarity. When listening through those live recording tracks and their corresponding studio versions, it turns out that there are significant structural differences between them. Usually the live recording has an extended section of instrumental improvisation in the beginning of the track that is musically very dissimilar to the original song version. These improvised segments can have a duration up to 90 seconds and thus create a major alignment mismatch between the recognizable parts of the tracks.

While the focus of this study is in live music song identification, a comparison between the proposed system and the CSI-methods from the literature can still be made. The obtained results with Covers80-dataset were clearly outperformed by the reported results of the reference implementation ( $P@10 = 0.094$ ,  $MR1 = 3.85$  and  $MAP = 0.909$ ) [16]. This raises a question about the competence of the proposed system given that similar approaches and data were used in both of the compared methods. The lower performance could be explained by the different choices made in the design of the model architecture. The differences between the used hyperparameters can also impact the performance to some extent. The fact that a significant proportion of the training data was unavailable and discarded in this study must have affected the results as well.

## 5.1 Feature Analysis

In order to better understand the performance of the system it is useful to examine the properties of the data and the types of input examples the model needs to deal with. Examples of input features from two versions of the same song are presented in Figure 5.2. The upper CQ-spectrogram in the figure contains the original studio version of the song while the lower CQ-spectrogram contains a live version of it. Both tracks have roughly the same instrumentation and are played by the same group of musicians. The song arrangements between the two recordings are also very close to each other which can be seen as structural similarities in the time-frequency domain. For instance, on both tracks there is a simple melody being played by a piano during the first five seconds of the song. The entering of a drum beat is also clearly visible on both tracks as it creates the sharp vertical patterns starting from around the 27th second. Moreover, by looking only at the average energy levels along the two time-frequency spectra it is possible to see where the song parts change from one section to another. Note that it is entirely song dependent whether the different song parts are as easily recognizable as in the given example.

In western popular music it is common that songs are divided into repetitive



**Figure 5.2** A Comparison of two CQ-spectrograms of the same song. The upper panel contains the original studio version of "Spread Your Wings" by "Queen", while the lower panel is from a live recording of the song performed by the same band. The number of frequency bins per octave is 12, the minimum frequency is 32.7 Hz and the window length as well as the hop size is 6592 samples.

sections. This means that two songs of any music genre that are musically dissimilar can share the same song structure and thus appear to have similar spectral features. The ability to determine the structure of a song is definitely useful when measuring the similarity of songs, but it should not suffice as a proof of similarity in itself. More important song properties in terms of song identification include the composition of the melodic patterns and their rhythms. The lyrics of a song are also important, but it is undeniable that successive pitches and their timing are what make songs unique and recognizable. Other elements that can be characteristic to a song include instrumentation, arrangement, tempo and the song key. However, these properties can be altered heavily and the resulted version will still present the musical piece with the corresponding melodies.

The importance of melodies has been taken into account in the proposed method through the design choices made in the feature extraction and feature representation stages. The use of CQ-spectrograms as the input features and the use of deep sequences allow the model to recognize melodic patterns at multiple levels of abstraction. In addition to note onsets, offsets and note intensities it is also important for the model to be able to recognize simultaneously played notes. This is because songs can sometimes rely more on chord sequences rather than clearly distinguishable lead melodies. However, the ability to separate all harmonic components at any given point also means that any unique song melodies are likely to overlap with less important notes played by other instruments. It can therefore be very difficult

to detect the exact melody patterns in a song when there are multiple instruments present at the same time. A good example of this can be seen in Figure 5.2. The model may confuse songs with similar melodies or chord progressions whenever there are other instruments playing the same note frequencies as the melody. Alternatively, non-existing melody patterns can be detected by concatenating individual notes of different instruments to form a sequence of notes that in reality are not perceived by a human listener.

## 5.2 Future improvements

There are several improvements that can be made to increase the system performance. First of all, the training data should be replaced or combined with a large collection of live music data. The song examples that were used in the training of the model differ from live music in many ways, so it is unreasonable to expect the system to master the task when it is trained with one type of data but evaluated with other. Getting more data is not always possible however. The amount of training data could still be increased by using various data augmentation techniques. For instance, instead of linking each cover song only with their original version, it is possible to treat each cover version as the original studio version and link the alternative song versions with each other. With negative examples of class 0 however the number of possible song combinations is measured in billions so only a fraction of it can be used due to limited computational resources.

Other data augmentation methods include feature extraction from different parts of the songs, mixing the songs with white noise or audio events such as crowd cheering or instrument sounds, manipulation of the song tracks by altering the song key and tempo, using different room responses and using both of the stereo channels of each song track separately. The extraction of multiple CQ-spectrograms with varying time alignments would most likely make the model more robust to major alignment mismatches. Also, making use of the separate stereo channels in each song could increase the diversity of the training data. This is due to the fact that different panning settings are often set to different instruments so splitting the channels would cause the sounds in the opposite stereo channel to become attenuated or entirely inaudible. The rest of the above mentioned data augmentation methods are much more laborious to carry out as they require additional signal processing and manipulation of the audio tracks. Apart from minor tempo fluctuations, song key changes and extra reverberation, artificially generated data might not necessarily even improve the model accuracy. This is because the simulations would have to be realistic enough to resemble the corresponding real life phenomena in order to be beneficial.

Song properties that are currently not exploited in the input features include

instrumentation and song lyrics. Although it is possible to recognize instruments based on CQ-transforms, the model should ideally treat all instruments similarly as it is more important to know what notes are being played instead of identifying what instruments are generating the sounds. An ability to detect the lyrics of the song from the audio tracks would allow comparing the songs in an entirely new feature domain. The role of song lyrics can be of great importance especially when dealing with less melodic music such as rap or certain sub-categories of metal music. Then again, the lyrics of a song can be sang in different languages and same words and sentences are often used in many songs which increases the complexity of the task even further.

Hyperparameter tuning can affect greatly in the system performance regardless of the amount of live music data used in the training. Some examples of tunable parameters are the learning rate of the optimizer, learning rate scheduler, the number of batch normalization and dropout-layers and the placement of these layers. In addition to model parameters, there are also configurations in the feature extraction stage that matter, such as the length of the input audio segments or the number of frequency bins used in CQ spectrograms. The effects of different feature extraction parameters remain unknown, except for the standardization of the features which resulted in faster convergence and better results.

Using an alternative loss function during the training stage could also improve the overall results. The problem with BCE-loss is that it is blind to relative differences between songs. This is because of the rough simplification of considering all the positive and negative examples equally similar or dissimilar, while in reality some songs are more similar in terms of musical properties than others. From a metric learning point of view, similar versions of the same song should be mapped onto nearby locations in an embedding space of songs. Triplet Loss and Lifted Structure Loss are examples of loss functions that could take this relative separation into account.

## 6 Conclusions

In this paper the topic of live music song identification was studied with the goal of automatizing the task by developing modern deep learning-based methods. A proposal of an information retrieval system was presented that, given a live recording of a song as an input query, is able to identify the original song version performed in the live recording and retrieve the identified song from a music database. Related literature was first studied as part of the research to benchmark the latest methods used in the task of cover song identification. Theoretical background about the topics applied in the system implementation was then presented, followed by a description of the system design. The theoretical background covered an introduction to the concepts of information retrieval and similarity learning as well as a comprehensive explanation about siamese convolutional neural networks and their training procedure. Public online music datasets were collected to enable the data driven development of the system. A manually collected custom live music dataset of live recordings and their corresponding studio versions was also collected. Experiments with the live music data were carried out to evaluate the performance of the system. The findings of the study were concluded in a discussion regarding the results of the experiments and how to further improve the system in the future.

The proposed system is comprised of a siamese convolutional neural network-based model and a database of studio versions of songs. The model makes use of cross-similarity matrices of multi-level deep sequences to measure the musical similarity between given music tracks. For each query entered to the system, a similarity score is produced for every song in the database. The song with the highest similarity score is assumed as the song being performed in the live recording. The system is able to identify 81,1% of the test queries after being trained with cover songs and evaluated with the custom live music dataset. The results of the experiments are promising considering the nature of the training data that lacks many properties of live music. However, the model tends to struggle with live recordings that have major structural differences with respect to their original song version. Further research is therefore needed to improve the detection performance and the reliability of the system. In overall the proposed method has lots of potential, and if developed further, it can help the music industry in streamlining the outdated royalty compensation practices of live music.

## References

- [1] *Advanced Topics in Information Retrieval*. eng. 1st ed. 2011. The Information Retrieval Series, 33. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. ISBN: 3-642-20946-7.
- [2] Syed Muhammad Anwar et al. “Medical Image Analysis using Convolutional Neural Networks: A Review”. eng. In: *Journal of Medical Systems* 42.11 (2018), pp. 226–13. ISSN: 0148-5598.
- [3] Andy Baio. “The tangled issue of cover song copyright on YouTube”. In: *Wired* (May 3, 2012). URL: <https://www.wired.co.uk/article/cover-song-licensing-on-youtube>.
- [4] J. C BROWN. “Calculation of a constant Q spectral transform”. eng. In: *The Journal of the Acoustical Society of America* 89 (1991). ISSN: 0001-4966.
- [5] Kang Cai, Deshun Yang, and Xiaou Chen. “Cross-Similarity Measurement of Music Sections: A Framework for Large-scale Cover Song Identification”. In: *Advances in Intelligent Information Hiding and Multimedia Signal Processing*. Vol. 63. Smart Innovation, Systems and Technologies. Springer International Publishing, 2016. ISBN: 9783319502083.
- [6] Sungkyun Chang et al. “Audio Cover Song Identification using Convolutional Neural Network”. eng. In: *arXiv.org* (2020). ISSN: 2331-8422.
- [7] Xingjian Du et al. “Bytecover: Cover Song Identification Via Multi-Loss Training”. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021. ISBN: 9781728176055.
- [8] Xingjian Du et al. “Bytecover2: Towards dimensionality reduction of latent embedding for efficient cover song identification”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2022, pp. 616–620.
- [9] D. P. W. Ellis. *The "covers80" cover song data set*. Accessed: 2022-06-21. 2007. URL: <http://labrosa.ee.columbia.edu/projects/coversongs/covers80/>.
- [10] Kunihiro Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. eng. In: *Biological Cybernetics* 36.4 (1980), pp. 193–202. ISSN: 0340-1200.

- [11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323.
- [12] Yoav Goldberg. *Neural Network Methods in Natural Language Processing*. eng. 1st ed. Vol. 37. Synthesis Lectures on Human Language Technologies 1. Morgan & Claypool Publishers, 2017, pp. 1–311. ISBN: 9781681732350.
- [13] *How Content ID works*. YouTube Help, Accessed: 2022-06-21. Google. 2022. URL: <https://support.google.com/youtube/answer/2797370?hl=en>.
- [14] D. H. Hubel and T. N. Wiesel. “Receptive fields of single neurones in the cat’s striate cortex”. eng. In: *The Journal of Physiology* 148.3 (1959), pp. 574–591. ISSN: 0022-3751.
- [15] Kazumasa Ishikura, Aiko Uemura, and Jiro Katto. “Live Version Identification with Audio Scene Detection”. In: *MultiMedia Modeling*. Lecture Notes in Computer Science. Springer International Publishing, 2015. ISBN: 3319144448.
- [16] Chaoya Jiang, Deshun Yang, and Xiaou Chen. “Similarity Learning For Cover Song Identification Using Cross-Similarity Matrices of Multi-Level Deep Sequences”. eng. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 26–30. ISBN: 9781509066315.
- [17] Diederik P. Kingma and Jimmy Lei Ba. “Adam: A method for stochastic optimization”. eng. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. 2015.
- [18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: <https://doi.org/10.1145/3065386>.
- [19] Bastian Leibe et al. “Gated Siamese Convolutional Neural Network Architecture for Human Re-identification”. In: *Computer Vision - ECCV 2016*. Lecture Notes in Computer Science. Springer International Publishing AG, 2016. ISBN: 9783319464831.
- [20] Guangzhe Liu, Ke Zhang, and Meibo Lv. “ASKs: Convolution with any-shape kernels for efficient neural networks”. eng. In: *Neurocomputing (Amsterdam)* 446 (2021), pp. 32–49. ISSN: 0925-2312.

- [21] Teresa Longjam, Dakshina Ranjan Kisku, and Phalguni Gupta. “Writer independent handwritten signature verification on multi-scripted signatures using hybrid CNN-BiLSTM: A novel approach”. eng. In: *Expert systems with applications* 214 (2023). ISSN: 0957-4174.
- [22] *Music Information Retrieval Evaluation eXchange (MIREX)*. Accessed: 2022-06-21. MultiMedia LLC. 2022. URL: [https://www.music-ir.org/mirex/wiki/2021:Audio%7B%5C\\_%7DCover%7B%5C\\_%7DSong%7B%5C\\_%7DIdentification](https://www.music-ir.org/mirex/wiki/2021:Audio%7B%5C_%7DCover%7B%5C_%7DSong%7B%5C_%7DIdentification).
- [23] NovaFrost. *SHS100K*. GitHub repository, Accessed: 2022-06-21. 2017. URL: <https://github.com/charlespwd/project-title>.
- [24] *PIKI Libraries*. Accessed: 2022-06-21. Finna. 2021. URL: <https://piki.finna.fi/>.
- [25] Atoany Nazareth Fierro Radilla and Karina Ruby Perez Daniel. “Siamese convolutional neural network for ASL alphabet recognition”. In: *Computacion y Sistemas* (2020). ISSN: 1405-5546.
- [26] Zafar Rafii, Bob Coover, and Jinyu Han. “An audio fingerprinting system for live version identification using image processing techniques”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014. ISBN: 9781479928927.
- [27] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. “On the Convergence of Adam and Beyond”. In: *International Conference on Learning Representations*. 2018.
- [28] Thomas D Rossing et al. *Springer Handbook of Acoustics*. Springer Handbooks. Springer, 2007.
- [29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *Nature (London)* 323 (1986). ISSN: 0028-0836.
- [30] Diego F. Silva et al. “SiMPLe: Assessing music similarity using subsequences joins”. In: *International Society for Music Information Retrieval*. 2016. ISBN: 0692755063.
- [31] Diego Furtado Silva, Felipe Falcao, and Nazareno Andrade. “Summarizing and Comparing Music Data and Its Application on Cover Song Identification.” In: *International Society for Music Information Retrieval*. 2018.
- [32] I Sonata et al. “Autonomous car using CNN deep learning algorithm”. eng. In: *Journal of Physics. Conference Series* 1869.1 (2021), pp. 12071–. ISSN: 1742-6588.

- [33] Ilya Sutskever et al. “On The Importance of Initialization and Momentum in Deep Learning”. eng. In: *30th International Conference on Machine Learning, ICML 2013*. 3. 2013, pp. 2176–2184.
- [34] T. J Tsai, Thomas Pratzlich, and Meinard Muller. “Known-Artist Live Song Identification Using Audio Hashprints”. In: *IEEE Transactions on Multimedia* 19 (2017). ISSN: 1520-9210.
- [35] Avery Wang et al. “An industrial strength audio search algorithm.” In: *ISMIR*. 2003, pp. 7–13.
- [36] Hao Wang et al. “CosFace: Large Margin Cosine Loss for Deep Face Recognition”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018.
- [37] Zhesong Yu et al. “Learning a Representation for Cover Song Identification Using Convolutional Neural Network”. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020. ISBN: 9781509066315.
- [38] Han Zhang et al. “Visual tracking using Siamese convolutional neural network with region proposal and domain specific updating”. eng. In: *Neurocomputing (Amsterdam)* 275 (2018), pp. 2645–2655. ISSN: 0925-2312.

## APPENDIX Custom Live Music Dataset

The following table contains a list of songs that were used as studio versions in the Custom Live Music Dataset. The Number of available live versions per song is also presented in the table. The performing artists, albums and song names are sorted in an alphabetical order, respectively.

| Artist name     | Album name                    | Song title                      | N live |
|-----------------|-------------------------------|---------------------------------|--------|
| ABC             | The Lexicon Of Love           | 4 Ever 2 Gether                 | 1      |
| ABC             | The Lexicon Of Love           | All Of My Heart                 | 1      |
| ABC             | The Lexicon Of Love           | Alphabet Soup                   | 1      |
| ABC             | The Lexicon Of Love           | Date Stamp                      | 1      |
| ABC             | The Lexicon Of Love           | Many Happy Returns              | 1      |
| ABC             | The Lexicon Of Love           | Overture                        | 1      |
| ABC             | The Lexicon Of Love           | Poison Arrow                    | 1      |
| ABC             | The Lexicon Of Love           | Show Me                         | 1      |
| ABC             | The Lexicon Of Love           | Tears Are Not Enough            | 1      |
| ABC             | The Lexicon Of Love           | The Look Of Love                | 1      |
| ABC             | The Lexicon Of Love           | Valentine's Day                 | 1      |
| Allen Toussaint | Everythibg I Do Gonh Be Funky | Fortune Teller                  | 1      |
| Atomic Rooster  | Death Walks Behind You        | Death Walks Behind You          | 1      |
| Atomic Rooster  | Death Walks Behind You        | Tomorrow Night                  | 1      |
| Black Sabbath   | Born Again                    | Digital Bitch                   | 1      |
| Black Sabbath   | Born Again                    | Hot Line                        | 1      |
| Black Sabbath   | Born Again                    | The Dark                        | 1      |
| Black Sabbath   | Born Again                    | Zero The Hero                   | 1      |
| Black Sabbath   | Mob Rules                     | Country Girl                    | 1      |
| Black Sabbath   | Mob Rules                     | E5150                           | 2      |
| Black Sabbath   | Mob Rules                     | Slipping Away                   | 1      |
| Black Sabbath   | Mob Rules                     | The Mob Rules                   | 2      |
| Black Sabbath   | Mob Rules                     | Voodoo                          | 2      |
| Black Sabbath   | Seventh Star                  | Danger Zone                     | 1      |
| Black Sabbath   | Seventh Star                  | Seventh Star                    | 1      |
| Bob Marley      | Kaya                          | Easy Skanking                   | 1      |
| Bob Marley      | Kaya                          | Is This Love                    | 1      |
| Bob Marley      | Legend                        | Exodus                          | 1      |
| Bob Marley      | Legend                        | Jamming                         | 1      |
| Bob Marley      | Legend                        | No Woman No Cry                 | 4      |
| Bob Marley      | Rastaman Vibration            | Positive Vibration              | 1      |
| Bob Marley      | Rastaman Vibration            | Rat Race                        | 1      |
| Bob Marley      | Rastaman Vibration            | Roots, Rock, Reggae             | 1      |
| Bob Marley      | Rastaman Vibration            | Want More                       | 1      |
| Cream           | Disraeli Gears                | Outside Woman Blues             | 1      |
| Cream           | Disraeli Gears                | SWLABR                          | 1      |
| Cream           | Disraeli Gears                | Strange Brew                    | 1      |
| Cream           | Disraeli Gears                | Take It Back                    | 1      |
| Cream           | Disraeli Gears                | Tales Of Brave Ulysses          | 1      |
| Cream           | Disraeli Gears                | We're Going Wrong               | 1      |
| Elvis Costello  | This Years Model              | (I Don't Want To Go To) Chelsea | 1      |
| Elvis Costello  | This Years Model              | Hand In Hand                    | 1      |
| Elvis Costello  | This Years Model              | Lipstick Vogue                  | 1      |
| Elvis Costello  | This Years Model              | Little Triggers                 | 1      |
| Elvis Costello  | This Years Model              | No Action                       | 1      |
| Elvis Costello  | This Years Model              | Pump It Up                      | 1      |
| Elvis Costello  | This Years Model              | Radio, Radio                    | 1      |

| Artist name      | Album name                        | Song title                       | N live |
|------------------|-----------------------------------|----------------------------------|--------|
| Elvis Costello   | This Years Model                  | The Beat                         | 1      |
| Elvis Costello   | This Years Model                  | You Belong To Me                 | 1      |
| Faith No More    | Angel Dust                        | A Small Victory                  | 1      |
| Faith No More    | Angel Dust                        | As The Worm Turns                | 2      |
| Faith No More    | Angel Dust                        | Be Aggressive                    | 1      |
| Faith No More    | Angel Dust                        | Easy                             | 1      |
| Faith No More    | Angel Dust                        | Kindergarten                     | 1      |
| Faith No More    | Angel Dust                        | Midlife Crisis                   | 1      |
| Faith No More    | Angel Dust                        | RV                               | 1      |
| Faith No More    | The Album Of The Year             | Collision                        | 1      |
| Faith No More    | The Real Thing                    | Surprise! You're Dead!           | 1      |
| Faith No More    | The Real Thing                    | Underwater Love                  | 1      |
| Faith No More    | The Real Thing                    | War Pigs                         | 6      |
| Faith No More    | The Works                         | Falling To Pieces                | 1      |
| Faith No More    | The Works                         | We Care A Lot                    | 2      |
| Greatful Dead    | The Greatful Dead                 | Beat It On Down The Line         | 1      |
| Greatful Dead    | The Greatful Dead                 | Cold Rain And Snow               | 1      |
| Greatful Dead    | The Greatful Dead                 | Cream Puff War                   | 1      |
| Greatful Dead    | The Greatful Dead                 | New, New Minglewood Blues        | 1      |
| Greatful Dead    | The Greatful Dead                 | Sittin' On Top Of The World      | 1      |
| Greatful Dead    | The Greatful Dead                 | Viola Lee Blues                  | 1      |
| Hound Dog Taylor | Beware of the Dog!                | Give Me Back My Wig              | 1      |
| Hound Dog Taylor | Beware of the Dog!                | Rock Me                          | 1      |
| Hound Dog Taylor | Beware of the Dog!                | The Sun Is Shining               | 1      |
| Jack Garratt     | Phase                             | Water                            | 1      |
| Jellyfish        | Spilt Milk                        | Joining A Fan Club               | 1      |
| Jellyfish        | Spilt Milk                        | The Ghost At Number One          | 1      |
| John Coltrane    | A Love Supreme                    | Acknowledgement                  | 2      |
| John Coltrane    | A Love Supreme                    | Psalm                            | 2      |
| John Coltrane    | A Love Supreme                    | Pursuance                        | 2      |
| John Coltrane    | A Love Supreme                    | Resolution                       | 2      |
| John Martyn      | One World                         | Certain Surprise                 | 1      |
| John Martyn      | One World                         | Couldn't Love You More           | 1      |
| John Martyn      | One World                         | Dealer                           | 1      |
| John Martyn      | One World                         | One World                        | 1      |
| John Martyn      | One World                         | Small Hours                      | 1      |
| Judas Priest     | Defenders Of The Faith            | Defenders Of The Faith           | 1      |
| Judas Priest     | Defenders Of The Faith            | Heavy Duty                       | 1      |
| Judas Priest     | Defenders Of The Faith            | Jawbreaker                       | 1      |
| Judas Priest     | Defenders Of The Faith            | Love Bites                       | 1      |
| Judas Priest     | Defenders Of The Faith            | Night Comes Down                 | 1      |
| Judas Priest     | Defenders Of The Faith            | Rock Hard Ride Free              | 1      |
| Judas Priest     | Defenders Of The Faith            | Some Heads Are Gonna Roll        | 1      |
| Judas Priest     | Defenders Of The Faith            | The Sentinel                     | 1      |
| Led Zeppelin     | Led Zeppelin: 2-CD Deluxe Edition | Dazed and Confused               | 1      |
| Led Zeppelin     | Led Zeppelin: 2-CD Deluxe Edition | How Many More Times              | 1      |
| Led Zeppelin     | Led Zeppelin: 2-CD Deluxe Edition | I Can't Quit You Baby            | 1      |
| Led Zeppelin     | Led Zeppelin: 2-CD Deluxe Edition | You Shook Me                     | 1      |
| Lenny Kravitz    | Mama Said                         | Always On The Run                | 2      |
| Lenny Kravitz    | Mama Said                         | Fields Of Joy                    | 1      |
| Lenny Kravitz    | Mama Said                         | More Than Anything In This World | 1      |
| Lenny Kravitz    | Mama Said                         | Stand By My Woman                | 1      |
| Lenny Kravitz    | Mama Said                         | Stop Draggin' Around             | 2      |

| Artist name     | Album name              | Song title                  | N live |
|-----------------|-------------------------|-----------------------------|--------|
| Lenny Kravitz   | Mama Said               | What The ... Are We Saying? | 1      |
| Little Mix      | Glory Days              | Touch                       | 1      |
| Meat Puppets    | Meat Puppets II         | Lake Of Fire                | 1      |
| Meat Puppets    | Meat Puppets II         | Plateau                     | 1      |
| Miles Davis     | Tutu                    | Portia                      | 1      |
| Miles Davis     | Tutu                    | Splatch                     | 1      |
| Monochrome Set  | Eligible bachelors      | Fallout                     | 1      |
| Monochrome Set  | Eligible bachelors      | Fun For All The Family      | 1      |
| Monster Magnet  | Powertrip               | Baby Götterdämmerung        | 1      |
| Monster Magnet  | Powertrip               | Bummer                      | 1      |
| Monster Magnet  | Powertrip               | Space Lord                  | 1      |
| Monster Magnet  | Powertrip               | Temple Of Your Dreams       | 1      |
| Monster Magnet  | Superjudge              | Dinosaur Vacume             | 1      |
| Monster Magnet  | Superjudge              | Evil                        | 1      |
| Monster Magnet  | Superjudge              | Superjudge                  | 1      |
| Monster Magnet  | Superjudge              | Twin Earth                  | 1      |
| Nirvana         | In Utero                | All Apologies               | 2      |
| Nirvana         | In Utero                | Dumb                        | 2      |
| Nirvana         | In Utero                | Tourette's                  | 1      |
| Nirvana         | Nevermind               | Breed                       | 1      |
| Nirvana         | Nevermind               | Come As You Are             | 2      |
| Nirvana         | Nevermind               | D-7                         | 1      |
| Nirvana         | Nevermind               | Drain You                   | 3      |
| Nirvana         | Nevermind               | In Bloom                    | 1      |
| Nirvana         | Nevermind               | Lithium                     | 1      |
| Nirvana         | Nevermind               | Lounge Act                  | 1      |
| Nirvana         | Nevermind               | On A Plain                  | 2      |
| Nirvana         | Nevermind               | Polly                       | 3      |
| Nirvana         | Nevermind               | Smells Like Teen Spirit     | 1      |
| Nirvana         | Nevermind               | Something In The Way        | 2      |
| Nirvana         | Nevermind               | Stay Away                   | 1      |
| Nirvana         | Nevermind               | Territorial Pissings        | 1      |
| Paloma Faith    | A Perfect Contradiction | Can't Rely On You           | 1      |
| Paloma Faith    | A Perfect Contradiction | Trouble With My Baby        | 1      |
| Paul Simon      | Stranger To Stranger    | Wristband                   | 1      |
| Phil Collins    | Hello, I Must Be Going! | I Cannot Believe It's True  | 1      |
| Phil Collins    | Hello, I Must Be Going! | I Don't Care Anymore        | 1      |
| Phil Collins    | Hello, I Must Be Going! | It Don't Matter To Me       | 1      |
| Phil Collins    | Hello, I Must Be Going! | Like China                  | 1      |
| Phil Collins    | Hello, I Must Be Going! | The West Side               | 1      |
| Phil Collins    | Hello, I Must Be Going! | Thru These Walls            | 1      |
| Phil Collins    | Hello, I Must Be Going! | You Can't Hurry Love        | 1      |
| Pierce The Veil | Misadventures           | "Floral & Fading"           | 1      |
| Pierce The Veil | Misadventures           | Today I Saw The Whole World | 1      |
| Queen           | Flash Gordon            | Flash                       | 1      |
| Queen           | Flash Gordon            | The Hero                    | 1      |
| Queen           | Jazz                    | Dreamers Ball               | 1      |
| Queen           | Jazz                    | Let Me Entertain You        | 1      |
| Queen           | News of The World       | My Melancholy Blues         | 1      |
| Queen           | News of The World       | Sheer Heart Attack          | 1      |
| Queen           | News of The World       | Spread Your Wings           | 1      |
| Queen           | The Game                | Dragon Attack               | 1      |
| Queen           | The Game                | Save Me                     | 1      |
| R.E.M.          | Green                   | Get Up                      | 4      |
| R.E.M.          | Green                   | Orange Crush                | 4      |
| R.E.M.          | Green                   | Pop Song 89                 | 4      |
| R.E.M.          | Green                   | Stand                       | 2      |
| R.E.M.          | Green                   | Turn You Inside-Out         | 2      |

| Artist name | Album name                                      | Song title                          | N live |
|-------------|---|-------------------------------------|--------|
| R.E.M.      | Green   | World Leader Pretend                | 2      |
| R.E.M.      | Green   | You Are The Everything              | 2      |
| R.E.M.      | Murmur  | 9-9                                 | 3      |
| R.E.M.      | Murmur  | Catapult                            | 1      |
| R.E.M.      | Murmur  | Laughing                            | 1      |
| R.E.M.      | Murmur  | Pilgrimage                          | 1      |
| R.E.M.      | Murmur  | Radio Free Europe                   | 3      |
| R.E.M.      | Murmur  | Sitting Still                       | 2      |
| R.E.M.      | Murmur  | Talk About The Passion              | 2      |
| R.E.M.      | Murmur  | We Walk                             | 1      |
| R.E.M.      | Murmur  | West Of The Fields                  | 2      |
| R.E.M.      | Out Of Time                                     | Belong                              | 2      |
| R.E.M.      | Out Of Time                                     | Country Feedback                    | 3      |
| R.E.M.      | Out Of Time                                     | Endgame                             | 1      |
| R.E.M.      | Out Of Time                                     | Half A World Away                   | 3      |
| R.E.M.      | Out Of Time                                     | Losing My Religion                  | 8      |
| R.E.M.      | Out Of Time                                     | Low                                 | 2      |
| R.E.M.      | Out Of Time                                     | Radio Song                          | 2      |
| R.E.M.      | Reckoning                                       | (Don't Go Back To) Rockville        | 3      |
| R.E.M.      | Reckoning                                       | 7 Chinese Bros                      | 3      |
| R.E.M.      | Reckoning                                       | Harborcoat                          | 4      |
| R.E.M.      | Reckoning                                       | Letter Never Sent                   | 3      |
| R.E.M.      | Reckoning                                       | Little America                      | 3      |
| R.E.M.      | Reckoning                                       | Pretty Persuasion                   | 5      |
| R.E.M.      | Reckoning                                       | Second Guessing                     | 4      |
| R.E.M.      | Reckoning                                       | So. Central Rain                    | 3      |
| Ramones     | Leave Home                                      | California Sun                      | 2      |
| Ramones     | Leave Home                                      | Glad To See You Go                  | 2      |
| Ramones     | Leave Home                                      | I Remember You                      | 2      |
| Ramones     | Ramones   | "53rd & 3rd"                        | 2      |
| Ramones     | Ramones   | Beat On The Brat                    | 2      |
| Ramones     | Ramones   | Blitzkrieg Bop                      | 2      |
| Ramones     | Ramones   | Chain Saw                           | 2      |
| Ramones     | Ramones   | Havana Affair                       | 2      |
| Ramones     | Ramones   | I Don't Wanna Walk Around With You  | 2      |
| Ramones     | Ramones   | I Wanna Be Your Boyfriend           | 2      |
| Ramones     | Ramones   | Judy Is A Punk                      | 2      |
| Ramones     | Ramones   | Let's Dance                         | 2      |
| Ramones     | Ramones   | Listen To My Heart                  | 2      |
| Ramones     | Ramones   | Loudmouth                           | 2      |
| Ramones     | Ramones   | Now I Wanna Sniff Some Glue         | 2      |
| Ramones     | Ramones   | Today Your Love, Tomorrow The World | 2      |
| Sex Pistols | Never mind The Bollocks, Here's The Sex Pistols | Anarchy In The U.K.                 | 6      |
| Sex Pistols | Never mind The Bollocks, Here's The Sex Pistols | EMI                                 | 1      |
| Sex Pistols | Never mind The Bollocks, Here's The Sex Pistols | God Save The Queen                  | 1      |
| Sex Pistols | Never mind The Bollocks, Here's The Sex Pistols | New York                            | 5      |
| Sex Pistols | Never mind The Bollocks, Here's The Sex Pistols | No Feelings                         | 5      |
| Sex Pistols | Never mind The Bollocks, Here's The Sex Pistols | No Fun                              | 6      |
| Sex Pistols | Never mind The Bollocks, Here's The Sex Pistols | Pretty Vacant                       | 5      |

| Artist name    | Album name                                      | Song title                        | N live |
|----------------|---|-----------------------------------|--------|
| Sex Pistols    | Never mind The Bollocks, Here's The Sex Pistols | Problems                          | 3      |
| Sex Pistols    | Never mind The Bollocks, Here's The Sex Pistols | Seventeen                         | 5      |
| Sex Pistols    | Never mind The Bollocks, Here's The Sex Pistols | Submission                        | 5      |
| Shawn Mendes   | Illuminate                                      | Mercy                             | 1      |
| Soundgardern   | Badmotorfinger                                  | Drawing Flies                     | 1      |
| Soundgardern   | Badmotorfinger                                  | Face Pollution                    | 1      |
| Soundgardern   | Badmotorfinger                                  | Jesus Christ Pose                 | 1      |
| Soundgardern   | Badmotorfinger                                  | Mind Riot                         | 1      |
| Soundgardern   | Badmotorfinger                                  | Outshined                         | 1      |
| Soundgardern   | Badmotorfinger                                  | Rusty Cage                        | 1      |
| Soundgardern   | Badmotorfinger                                  | Searching With My Good Eye Closed | 1      |
| The Cure       | Disintegration                                  | Closedown                         | 1      |
| The Cure       | Disintegration                                  | Disintegration                    | 1      |
| The Cure       | Disintegration                                  | Fascination Street                | 1      |
| The Cure       | Disintegration                                  | Homesick                          | 1      |
| The Cure       | Disintegration                                  | Last Dance                        | 1      |
| The Cure       | Disintegration                                  | Lovesong                          | 1      |
| The Cure       | Disintegration                                  | Lullaby                           | 1      |
| The Cure       | Disintegration                                  | Pictures Of You                   | 1      |
| The Cure       | Disintegration                                  | Plainsong                         | 1      |
| The Cure       | Disintegration                                  | Prayers For Rain                  | 1      |
| The Cure       | Disintegration                                  | The Same Deep Water As You        | 1      |
| The Cure       | Disintegration                                  | Untitled                          | 1      |
| The Cure       | Seventeen Seconds                               | A Forest                          | 1      |
| The Cure       | Seventeen Seconds                               | A Reflection                      | 1      |
| The Cure       | Seventeen Seconds                               | At Night                          | 1      |
| The Cure       | Seventeen Seconds                               | I Dig You                         | 1      |
| The Cure       | Seventeen Seconds                               | I'm A Cult Hero                   | 1      |
| The Cure       | Seventeen Seconds                               | In Your House                     | 1      |
| The Cure       | Seventeen Seconds                               | M                                 | 1      |
| The Cure       | Seventeen Seconds                               | Play For Today                    | 1      |
| The Cure       | Seventeen Seconds                               | Seventeen Seconds                 | 1      |
| The Cure       | Seventeen Seconds                               | The Final Sound                   | 1      |
| The Pretenders | Learning To Crawl                               | My City Was Gone                  | 1      |
| The Pretenders | Learning To Crawl                               | Time The Avenger                  | 1      |
| The Who        | The Who Hits 50!                                | Happy Jack                        | 1      |
| The Who        | The Who Hits 50!                                | I Can't Explain                   | 1      |
| The Who        | The Who Hits 50!                                | Magic Bus                         | 1      |
| The Who        | The Who Hits 50!                                | My Generation                     | 1      |
| The Who        | The Who Hits 50!                                | Pinball Wizard                    | 1      |
| The Who        | The Who Hits 50!                                | Substitute                        | 5      |
| Underworld     | Dubnobasswithmyheadman                          | Mmm Skyscraper I Love You         | 1      |
| Underworld     | Dubnobasswithmyheadman                          | Spoonman                          | 1      |
| Uriah Heep     | Look At Yourself                                | Look At Yourself                  | 1      |
| Uriah Heep     | Look At Yourself                                | What Should Be Done               | 1      |
| Uriah Heep     | Sweet Freedom                                   | Stealin'                          | 1      |
| Uriah Heep     | Sweet Freedom                                   | Sweet Freedom                     | 1      |
| Uriah Heep     | Very 'eavy ...very 'umble                       | Dreammare                         | 1      |
| Uriah Heep     | Very 'eavy ...very 'umble                       | Gypsy                             | 1      |
| Uriah Heep     | Wonderworld                                     | I Won't Mind                      | 1      |
| Uriah Heep     | Wonderworld                                     | So Tired                          | 1      |
| Volbeat        | Let's Boogie!                                   | Black Rose                        | 1      |
| Volbeat        | Let's Boogie!                                   | For Evigt                         | 1      |
| Volbeat        | Let's Boogie!                                   | Goodbye Forever                   | 1      |

| Artist name | Album name        | Song title                                   | N live |
|-------------|-------------------|--|--------|
| Volbeat     | Let's Boogie!     | Let It Burn                                  | 1      |
| Volbeat     | Let's Boogie!     | Seal The Deal                                | 1      |
| Volbeat     | Let's Boogie!     | Slaytan                                      | 1      |
| Volbeat     | Let's Boogie!     | The Devil's Bleeding Crown                   | 2      |
| Whiskeytown | Strangers Almanac | 16 Days                                      | 1      |
| Whiskeytown | Strangers Almanac | Avenues                                      | 1      |
| Whiskeytown | Strangers Almanac | Excuse Me While I Break My Own Heart Tonight | 1      |
| Whiskeytown | Strangers Almanac | Houses On The Hill                           | 1      |
| Whiskeytown | Strangers Almanac | Somebody Remembers The Rose                  | 2      |
| Whiskeytown | Strangers Almanac | Turn Around                                  | 1      |

**Table Appendix.1** List of songs in the Custom Live Music Dataset. "N live" stands for the number of live recordings of the corresponding song.