|  |
| --- |
| Imperial College London  Department of Electrical and Electronic Engineering |
| Final Year Project Report |
| Machine learning applied to timbral acoustic analysis |
| June 2021 |

|  |  |
| --- | --- |
| Student:  CID:  Course:  Project Supervisor:  Second Marker: | **Tom Perkins**  **01391025**  **4th Year Electronic and Information Eng.**  **Professor Patrick A. Naylor**  **Professor Athanassios Manikas** |

Abstract

This project investigates the analysis of musical instrument timbre as a perceptual task by using Convolutional Neural Networks (CNNs). The timbre of a musical instrument is what allows it to be distinguished sonically from other instruments, independently of pitch and volume. We have chosen to approach timbral analysis by framing the task as an identification or classification problem, as opposed to targeting a glossary of timbral descriptions such as “bright”, “mellow”, which is a common alternative approach. Neural Networks are readily adapted to handling such classification problems; in particular, we will focus on the binary classification problem of distinguishing between recordings of different types of acoustic pianos using timbre. This is tackled with CNNs by transforming waveforms into spectrograms, so that two-dimensional maps can be input to the network. This approach allows us to draw inspiration from the convolutional architectures commonly used in image analysis, while applying signal processing and musical-domain considerations.

Table of Contents

[1 Introduction 4](#_Toc74754968)

[2 Literature review of timbral analysis methods 5](#_Toc74754969)

[2.1 Conventional signal processing methods 5](#_Toc74754970)

[2.1.1 Inter-instrument and instrument type classification 5](#_Toc74754971)

[2.1.2 Intra-instrument classification 6](#_Toc74754972)

[2.2 Timbral analysis using neural networks 7](#_Toc74754973)

[2.2.1 Perceptual CNN standards 8](#_Toc74754974)

[2.2.2 CNNs applied to timbre analysis 9](#_Toc74754975)

[2.3 Research on related topics 10](#_Toc74754976)

[3 Background theory 11](#_Toc74754977)

[3.1 Signal processing features and theory for characterising timbre 11](#_Toc74754978)

[3.1.1 General background 11](#_Toc74754979)

[3.1.2 Temporal features 13](#_Toc74754980)

[3.1.3 Spectral features 14](#_Toc74754981)

[3.1.4 Spectro-temporal features 15](#_Toc74754982)

[3.1.5 Harmonic features 17](#_Toc74754983)

[3.1.6 Formant analysis and the source-filter model 19](#_Toc74754984)

[3.1.7 Cepstral features and the Mel scale 21](#_Toc74754985)

[3.2 Machine learning applied to timbral identification 25](#_Toc74754986)

[3.2.1 Non-neural network methods 25](#_Toc74754987)

[3.2.2 Neural Networks and Deep Learning methods 26](#_Toc74754988)

[4 System Specification 34](#_Toc74754989)

[4.1 Selection of input features: log-Mel spectrogram 34](#_Toc74754990)

[4.2 Problem specification: Binary classification of piano types using CNNs 35](#_Toc74754991)

[4.3 Datasets assembled for the task 36](#_Toc74754992)

[4.3.1 Assembly of the isolated-notes dataset using single-note piano samples 36](#_Toc74754993)

[4.3.2 Generation of the melody dataset: alternative dataset derived from single-note data 39](#_Toc74754994)

[5 Design and analysis 40](#_Toc74754995)

[5.1 Pre-processing Methods 40](#_Toc74754996)

[5.1.1 Processing of raw single-note waveforms 40](#_Toc74754997)

[5.1.2 Feature extraction: generation of Mel spectrograms 42](#_Toc74754998)

[5.1.3 Isolated-notes-specific spectrogram processing 43](#_Toc74754999)

[5.1.4 Melody-specific processing 44](#_Toc74755000)

[5.2 Dataset partitioning for classifier training and evaluation 46](#_Toc74755001)

[5.3 CNN Architecture and Training 48](#_Toc74755002)

[5.3.1 CNN architecture design 48](#_Toc74755003)

[5.3.2 The training process and hyperparameters 52](#_Toc74755004)

[6 Model selection and evaluation methodology 54](#_Toc74755005)

[6.1.1 Scoring the classifier 54](#_Toc74755006)

[6.1.2 Model selection and validating the classifier 56](#_Toc74755007)

[7 Implementation 60](#_Toc74755008)

[7.1 Software standards and toolkits 60](#_Toc74755009)

[7.2 Data-loading implementation 61](#_Toc74755010)

[7.3 System software structure 61](#_Toc74755011)

[8 Classification Results 63](#_Toc74755012)

[8.1 Model selections & training behaviour for each task 63](#_Toc74755013)

[8.1.1 Single-note classification 63](#_Toc74755014)

[8.1.2 Melody-based classification 65](#_Toc74755015)

[8.2 Comparison of cross-validation results across both tasks 67](#_Toc74755016)

[8.3 Held-out test set results 69](#_Toc74755017)

[8.3.1 Overall performance 69](#_Toc74755018)

[8.3.2 Per-instrument results 71](#_Toc74755019)

[Conclusions, Discussion and Further work 71](#_Toc74755020)

[Evaluation 71](#_Toc74755021)

[Limitations 72](#_Toc74755022)

[Potential for Further Work 73](#_Toc74755023)

[Appendix A – Review of isolated-note datasets 74](#_Toc74755024)

[References 75](#_Toc74755025)

# Introduction

Acoustical timbre is the quality of a sound that allows it to be “distinguished from other sounds at the same pitch and loudness” [1]. Thus, it is commonly defined by elimination, characterising all that does not correspond to pitch and volume in a sound. For the purposes of this project, we consider timbre as encapsulating the qualities characteristic to a musical sound, which allow it to be recognised as having been produced by a particular instrument.

Timbral analysis consists of automatically extracting information from a digital audio signal, in order to describe a sound source’s identifying timbral qualities. Timbre is a perceptual quality of sound, therefore it is subjective and can be complex to describe quantitatively. Therefore, accurately describing timbre in order to approximate the human ear’s fine differentiation abilities is a challenging problem not usually tackled by a single conventional signal processing method, such as spectral analysis, in isolation. The motivation for this project is to apply signal processing methods in combination with machine learning constructs to develop a system that can infer embeddings corresponding to timbral information, in order to distinguish between closely related musical sounds.

The timbre of a musical instrument can be affected by factors such as component materials, age, quality of construction, performance environment, and the articulation (or touch) of the player. For this reason, analysis of timbre may not only concern differentiating between instruments from different families, but also between different instances and types within a single family such as acoustic pianos, since these factors are rarely constant from one individual instrument to the next. While many research works have attempted to differentiate between different families of musical instruments by timbre (inter-instrument classification), few have focused on identifying the subtle variations that exist between different instances of the same type of instrument (intra-instrument classification). For this project, we focus our efforts on developing a system to detect the timbral differences between different types of acoustic piano, namely in order to classify upright and grand pianos. This type of fine differentiation task presents a relatively novel challenge, and can be tricky even for humans, as it requires a deep familiarity with the instruments.

The principal objective for the final system is to be able to distinguish between the two types of piano by using only an audio recording of the instrument as input. Another key aim for the classifier is to characterise timbre in a generalisable manner and in realistic scenarios, for example by identifying unseen examples recorded in different conditions than those seen in training.

In this report, we will first survey research that has been performed in the fields of timbral analysis and machine learning, especially research that applies machine learning to timbral classification of musical instruments. Then we will detail the background theory involved in timbral analysis, discussing both conventional signal processing features, and the machine learning methods studied and applied in the project. Next, we will describe the design considerations applied to the development the proposed system, including detail of the choices made at each stage: specification of the problem, data sourcing and processing, the decision to use Convolutional Neural Networks, their architecture, and the training process. In subsequent sections, we will describe the testing methods elaborated to evaluate the classifier, before reporting the results obtained for the final system in various scenarios. Finally, we will evaluate these results, present our conclusions, and ideas for potential further development.

# Literature review of timbral analysis methods

We subdivide the wide variety of timbral analysis research in the literature into approaches focusing mainly on timbral feature extraction via traditional signal processing methods, and methods which employ neural networks to learn the timbre of musical instruments for classification. Note that this is somewhat of a false dichotomy, since machine learning methods often inherently rely on signal processing theory and constructs, and usually depend on pre-processing of input features via signal processing. In this section, we will first give an overview of research on musical instrument timbral analysis performed with an emphasis on signal processing methods. We will then present a survey of more recent works which apply neural networks, specifically CNNs, to classifying the timbre of musical instruments, before detailing research that has been performed on related tasks in the field.

## Conventional signal processing methods

Many authors have tackled the problem of analysing timbre using signal processing descriptors, as well as the selection of an optimal set of these features for the purposes of identifying or differentiating instruments by their timbre. In order to gauge their timbral discrimination ability, authors usually use these features as input to simple machine learning systems such as K-Nearest Neighbour (KNN), Support Vector Machine (SVM) classifiers or Gaussian Mixture Models (GMM), but the emphasis of the works presented in this subsection remains on the feature selection and computation, as opposed to the optimisation of the downstream machine learning models.

### Inter-instrument and instrument type classification

First, we consider research aiming to discriminate between different instruments and broad families, as opposed to identifying the more subtle timbral variations between models, types or instances of the same instrument. One of the earlier such works on timbral instrument classification focuses on the steady-state part of individual isolated musical notes [2]. This approach narrows the analysis to the harmonic, or tonal, qualities of the sound as opposed to analysing the time evolution of the envelope and transient qualities over the course of a note. As a result, the author finds that discrimination between the considered orchestral instruments is achieved most effectively using spectral and harmonic features such as the spectral moments and the first two harmonic frequencies. The performance of the proposed system, which is a KNN-based classifier, also varies with the considered instrument, reporting much higher accuracy at identifying typically longer-sustained sounds, such as wind instruments, than shorter, more transient sounds such as plucked (*pizzicato*) violin. This shortcoming could be explained by the fact that the shorter sounds tend to be sonically recognisable primarily through their temporal envelope, specifically their harsh attack, as opposed to more sustained instrument timbres, which tend to contain richer tonal and harmonic information.

Another work which focuses on spectral and harmonic feature-based identification of instruments is [3], in which various classification methods (SVM, decision trees & Discriminant Analysis) are applied to identifying classical orchestra instruments from single-note recordings. Similarly to [2], the most representative features for classification are found to be the spectral centroid and the first partial’s energy; as well as inharmonicity, which as explained in Section 3.1.5 quantifies the prominence of harmonics – thus expressing the extent to which the sound is tonal or noise-like. As a result of incorporating this additional information along with purely tonal descriptors, the authors report reasonable success in identifying both sustained and transient sounds such as *pizzicati*, although once again the highest scores are reported for identification of wind instruments. In the research presented in [4], many of the same spectral and harmonic features are also applied to classifying a broader range of instruments, including a large corpus of non-western sounds. Beyond the tonal features, the authors also find the attack slope, which characterises the speed of the transient, to be among the most indicative features; along with descriptors of the spectral envelope. These findings indicate that the accurate timbral classification of a wide range of sounds benefits from a combination of spectral/harmonic and time-envelope analyses.

Other research works integrate cepstral features as well as spectral and temporal descriptors in order to classify musical instruments by timbre. For instance in [5], cepstral and time-autocorrelation coefficients are found to be instrumental in accurately differentiating between the timbres of four woodwind instruments using a GMM. Notably, the input to this model consists of extracts from single-instrument performances as opposed to single-note samples. The input information is therefore more complex in nature, but may contain more timbral cues than a single isolated note. This indicates that cepstral features, paired with temporal descriptors, are well-adapted to capturing complex information efficiently; especially since only a small number (10) of cepstral coefficients is used. Similarly, the method presented in [6] applies MFCCs computed at different time scales as well as the time-derivatives of MFCCs to identify instruments using an SVM classifier from excerpts of single-instrument performances. This approach aims to capture timbral features at different time resolutions, both considering the finer details within a single note’s envelope as well as on a larger scale for short monophonic musical phrases. The temporal scale of analysis for MFCCs in the context of musical instrument identification was previously examined in [7], a work which also takes on the task of classifying clips of real single-instrument performances using a large number of temporal, spectral, wavelet, and cepstral features.

### Intra-instrument classification

A small number of research works focus on differentiating between the timbre of different instances of the same instrument type using signal processing features; either aiming to identify playing techniques or variation in the sonic qualities of the instruments themselves. For instance, in [8], the author examines the timbre of Irish traditional flute playing in order to detect the articulation, phrasing and the model of flute captured in a given recording. The analysis of playing style corresponds to higher-level longer timescale timbral characteristics such as trills, while the single-note study of timbral variation between flutes made from different materials is most relevant to our purposes. Many pre-processing steps are applied for feature extraction and note onset detection: fine variations in timbre between instruments are quantified using spectral and harmonic peak analysis on the harmonically-stable (steady-state) portions of single note recordings, which tends to be effective for timbral characterisation of wind instruments as we have seen in [2].

In the same vein, [9] attempts a similar task by examining the timbre of violins from two different eras. A set of spectral, temporal, harmonic and envelope features are selected to help make this differentiation on sets of single-note and musical scale recordings. Dimensionality reduction (via a clustering method called t-distributed Stochastic Neighbour Embedding (t-SNE)) of the feature space allows for a projected 3-dimensional visualisation to be used to compare the timbral similarity between the instruments. This paper finds that analysing the steady state and decay phases of the violin note sounds allows for differentiation between the two classes of contemporary and historical violins. This differentiation is made on the strength of certain timbral features, determined using feature selection and ranking tests; in particular are selected the spectral distribution features (the spectral moments, roll-off and flatness measures), as well as the Spectral Flux and MFCCs. The relevance of the selected candidate timbral features is verified by inputting them to a SVM for classification between the two types of violins considered: “contemporary” and “historical”. Low classification error rates are reported especially when identifying single-note recordings of open strings, which are considered to have more complex harmonic content in the sustain and decay phases of the note envelope, allowing for finer differentiation between violins of different quality.

## Timbral analysis using neural networks

We now turn our attention to research on the subject of identifying musical instrument timbre using neural networks. Despite their inherent advantages over other machine learning methods in terms of their ability for generalisation to unseen data and training set noise insensitivity [10], fully-connected neural network models provided limited early success when applied to musical instrument classification compared to other classifiers such as K-Nearest-Neighbours [11]. However, as initially demonstrated on visual character recognition with the LeNet-5 architecture [12], CNNs are more naturally suited to higher dimensional inputs such as images or spectrograms (both of which are 2-dimensional vectors), compared to fully-connected neural networks. CNNs are therefore popular in the timbral analysis and classification literature thanks to their ability to make perceptual inferences. In order to operate on audio signals, these timbral networks are commonly input pre-processed spectrograms, a feature which CNNs are able to handle and learn from efficiently by their propensity to process large amounts of data in 2-dimensional features such as time-frequency maps. This approach allows for popular image processing CNN architecture paradigms to be applied to acoustic signal analysis, and we will review approaches in the literature that leverage this interesting method in this subsection.

### Perceptual CNN standards

To solve complex perceptual inference tasks such as image recognition, CNNs have gained popularity in the last decade as the standard machine learning approach which does not require manually engineered features to be extracted from the input images, since these are learned by the CNN. A pivotal development in this field was AlexNet [13], which won the 2012 ImageNet image classification contest [14]. This was a large-scale network which has set the precedent for using a large number of filters within each convolutional layer (convolution channels). Compared to LeNet-5 [12], AlexNet increased the width of each convolution’s receptive field within images, increasing the number of optimisable filter weights. For instance, the first convolutional layer in the network uses 11x11 filter kernels, which, paired with pixel-region pooling and strided operations helps the network achieve dimensionality reduction on the larger and more complex 224x224 input images. The resulting leap in complexity from 600,000 to 46 million total optimisable parameters in AlexNet compared to LeNet was enabled by leveraging hardware developments in graphics processing units (GPUs), which are better suited to the matrix operations involved in CNN training and inference [15].

This increase in CNN complexity was continued by the introduction of the VGG network [16], which ranked top in the ImageNet Challenge 2014, achieving higher performance over AlexNet. The main contribution of VGG was increasing the number of convolutional layers in the network and reducing the kernel size of each to only 3x3 pixels. This was informed by the authors’ analysis which found that a larger number of smaller convolution filters are more expressive than a sequence of fewer large-kernel convolutional layers. This informed the trend in recent years in deep convolutional neural networks (DNNs) towards “deep and narrow” architectures using large number of filters, each using small filters [17].

Further developments have aimed to reduce network complexity by simplifying architectures and reducing parameter counts in order to create more compact models while maintaining performance. This has been achieved through structural innovations such as residual connections, which add parallel paths allowing layer inputs to bypass convolutional blocks to be added to a layer’s output. These were introduced in the ResNet architecture [18], the authors aiming to use the residual connections to increase expressiveness by making the network function behave by default as an identity mapping from input to output, essentially recentring the potential class of models which can be learnt by a given CNN towards more natural functions (by biasing the models towards the identity function as a “starting point”) [19]. As a result, ResNet is considered as a state-of-the-art standard architecture for general perceptual tasks such as visual feature extraction, dimensionality reduction, and image recognition.

### CNNs applied to timbre analysis

One CNN-based timbral classification system [20] is used to identify the predominant instruments in recordings of multi-instrument mixtures, using a deep CNN made up exclusively of small 3x3 filters in each convolutional layer, separated periodically by max-pooling layers for dimensionality reduction from the input spectrograms via abstracted feature maps to the low-dimensional fully-connected output layers. Broadly speaking, this type of deep and narrow CNN architecture has been commonly applied across the literature in recent years. For instance, [21] improves upon this system by adding source separation of the instrument mixture as a pre-processing step, achieving improved instrument classification results on identifying jazz instruments with closely related timbres. Contributing to this method’s success, the authors also cite transfer learning as a good way of getting around limited training data. This is achieved by using a model pre-trained on a different, larger dataset as a starting point for the CNN to then learn more application-specific timbral mappings from the small targeted dataset of jazz instruments.

Another piece of research considering classification the most prominent instrument from recordings of pieces played by multi-instrument mixtures is [22], which also uses log-Mel spectrograms as the input feature map. This work focuses on how to design CNN architectures to effectively capture timbral information, using musically-informed intuitions such as the fact that timbre should be inferred independently from pitch, duration and volume. Given that the network's layers operate on the spectro-temporal domain, the optimal choice of dimensions in time (number of frames) and frequency (number of bins) of the convolutional layers is discussed. The experiments on sung phoneme classification and instrument classification lead the authors to conclude that the first layer of the CNN benefits from using a diversity of filter dimensions to capture different scales of time-frequency feature mappings. Additionally, max-pooling layers over the frequency dimension are used in order to reduce the effects of pitch, as it can be shown that max-pooling (as opposed to average pooling) mitigates the effect of shifting the input of a CNN layer on its output feature maps.

Other CNN-based methods using Mel spectrograms are applied to less challenging data in the form of isolated recordings of individual instruments, often restricted to playing a single note at a time. In [23], the authors apply this class of method to recognising classical instrument families using a variety of recording types, comparing the performance of CNNs trained on Mel spectrograms drawn from isolated notes, monophonic melodies, and polyphonic pieces. It is found that CNNs trained on one sort of data do not generalise well to classifying the same instruments from another type of recording: for instance, high accuracy is reported for a model trained and tested on single-note recordings, but this performance does not carry over to testing the same model on recordings of pieces played on the same set of instruments. This implies that in the paradigm of CNNs using Mel Spectrograms as input, models are sensitive to polyphony, and that the embeddings that allow them to differentiate between single note samples of instruments differ from those characterising timbre in recordings of a piece of music being played on those same instruments.

## Research on related topics

Besides musical instrument classification, a great deal of recent research has concerned applications of and tasks related to timbral analysis. These include:

* *Music Information Retrieval* (MIR) consists of automatic indexing of audio metadata. [24] gives an excellent overview of neural-network based approaches to MIR, many of which are also applicable to timbral analysis. A common MIR task is genre or style identification, which is achieved by similar feature-based means to instrument classification, but applied to learning the texture or other style cues of a full mix of music as opposed to that of an isolated instrument. A popular reference in the literature for this task is the approach presented in [25], where conventional signal processing timbral features are applied alongside features related to pitch and rhythm description for genre classification.
* *Source separation*, which aims to separate out audio corresponding to the different instruments in a mix. Notably, this task is tackled in [26] and [27], where the problem of overlapping harmonic partials (formants) between 2 instruments in a mix is mitigated using other timbral descriptors. In [26], the estimated spectral envelope of each instrument in the mixture is used to help separate the instruments from one another, while [27] aims to separate the sources by exploiting differences in their amplitude and frequency modulation characteristics (i.e. vibrato and tremolo playing effects respectively in musical terms).
* *Musical synthesis* and instrument *timbre transfer*: [28] uses a generative system (variational auto-encoder) to map musical instrument audio to a latent timbral space, a type of approach which the authors recently applied [29] to synthesising new sounds by selecting points in the space to transfer the timbre of one instrument to a different instrumental performance (e.g. between orchestral instruments and voice).
* *Speech recognition & diarization* (differentiation of who is speaking when): similarly to identifying variations between instruments, the timbral quality of the human voice is often used to help identify a speaker, which can be applied to diarization for dialogue transcription and voice authentication. Recent work [30] applies CNNs using Mel-spectrograms as input to the timbral classification of different voices on the basis of gender and age labels.

Another promising recent development relevant to timbral analysis is the integration of conventional signal processing elements, such as those presented in Section 3.1 for extraction of timbral features, into an end-to-end neural network architecture [31]. This allows signal processing functions to be used within a deep-learning framework, as opposed to being limited to use as pre-processing steps. Notably, qualitatively promising results for this system are demonstrated on timbre transfer from voice to violin, as well as on the decomposition of musical instrument sounds into noise-like and tonal components.

*TODO: Add detail on who would be interested in the proposed timbral classifier that can identify fine differences between different acoustic pianos, e.g.:*

* *For automatic indexing of audio libraries, e.g. on streaming services, by the type of instrument present in the audio recording.*
* *For automatic indexing of sample libraries for musicians, so that musicians can search for a specific timbral class of instrument such as upright or grand pianos.*
* *To help instrument manufacturers, specialists, or virtual instrument creators better understand which technical features of acoustic pianos are responsible for their characteristic timbre which makes them sound like a particular type of piano. This information could be used to tune the sound of a piano in order to modify its character.*

# Background theory

## Signal processing features and theory for characterising timbre

### General background

In this section, we present an overview of the most popular signal processing concepts, methods and features frequently applied in the literature to timbral analysis tasks. Understanding the role of these features, how to compute them and their interpretation will be instrumental in selecting a musically relevant and representative set of candidate features to use as pre-processed input to the timbral classifier. In the following discussion, sources [32] and [33] are referenced as comprehensive summaries containing more detailed definitions of each timbral feature and their computation, a subset of which are presented in the popular MPEG-7 standard for audio descriptors [34].

Digital audio signals being made up of samples recorded at a rate , features are often calculated over frames of length , where is the number of samples in a frame. Temporal features, which we will discuss in Section 3.1.2, are computed from the waveform in the time domain, within a given frame or over several frames.

Conversely, spectral features are drawn from the magnitude spectrum, in the frequency domain, as discussed in Section 3.1.3. Transformation into the frequency domain of an -sample signal is performed by applying the Discrete Fourier Transform (DFT), as follows (adapted from [35], slide 4):

This operation produces complex values in the frequency domain, each indexed by , which corresponds to a frequency Hz in normalised units (relative to the sampling frequency ). Frequency-domain analysis relies on the fact that musical sounds have periodic components, whose frequency distribution can be deduced from the spectrum . Some analysis methods aim to measure the extent to which the signal is periodic, as opposed to noise-like, in order to characterise whether the perceived timbre is more harmonic/pitched in the former case, or whether it is breathy in the latter case. We will discuss these approaches in sections 3.1.5 and 3.1.6.

In order to compute the DFT over a frame of analysis of finite length, the discrete Short-Term Fourier Transform (STFT) is applied, which produces the spectrum of the signal on a per-frame basis. This time-frequency representation allows for analysis of the magnitude of the frequency bins within a frame, as well as analysis of the evolution of the spectrum over consecutive frames. These methods yield the category of spectro-temporal features, which we will detail in Section 3.1.4.

Computing the STFT yields a complex value for each frequency index and at each time frame ending at sample index , as follows (adapted from [36], Eq. 4.68):

In , the STFT is taken over a frame of samples in the input signal , producing frequency bins, and is a discrete window function of length . A commonly used function for is the Hamming window (adapted from [37], slide 5):

This window function serves a similar purpose as others with bell-shaped frequency responses, such as the Blackman-Harris and Hanning windows, by limiting spectral leakage artifacts caused by the boundary effects of windowing when the analysis frame is not the exact length of a period.

Phase information is not usually considered for timbral analysis, as it is broadly assumed that the character of a musical sound can be inferred from its waveform amplitude and magnitude spectrum primarily [38]. Therefore the phase characteristics of the signal are not considered in our discussion, and from the STFT we retain the magnitude spectrum .

### Temporal features

For extraction of temporal features, which concern the time evolution of the waveform over the course of the sound segment, we assume the signal analysis is applied to a single, isolated tone representative of the sound. For musical instruments, this would correspond to a single note played in isolation, and recorded from its onset to finish.

Temporal envelope*(energy envelope)*

The envelope of a waveform is a smoothed version of the signal indicating the overall amplitude shape that the signal takes on over time. This can be achieved in its simplest form by taking the local average [33] or maximum of the waveform’s amplitude over a moving window, as demonstrated in [39].

#### Envelope attack, sustain and decay

* The ***attack time*** is defined as the period between the start of the sound until its maximum amplitude is reached [33]. Typically, the start of the attack is estimated by finding the time step at which a threshold (e.g. 10%) proportional to the amplitude’s maximum value over the considered sound is surpassed [32].
* The ***attack slope*** over the attack period further parametrises the speed of a sound’s rise, and is inferred from the average rate of increase of the waveform magnitude over the attack period [32].
* The ***steady-state***, or ***sustain*** period, corresponds to the phase after the attack during which the magnitude remains approximately constant near its maximum, and can be characterised by its length (sustain time).
* The ***decrease*** or ***decay***is characterised by the decrease slope, which can be calculated by estimating the rate at which the signal decays from the maximum-energy point [32].

The shape of the envelope characterises important timbral information relating to the articulation and form of a musical sound. For instance, a note played with *staccato* (“attacked”) articulation typically has a short envelope with a rapid rise (short attack time), as opposed to a note articulated as a swell, which will have a slower rise due to the note amplitude’s gradual increase initially. On a finer level, these envelope parameters depend not only on articulation (how the instrument is played), but also on the type of instrument and variations between different models of the same instrument, and have been shown experimentally to play an important role in humans’ perceptual ability to identify instruments [40].

#### Temporal centroid

The temporal centroid of a sound measures the time instant around which the energy of a sound is centred[33]. This is estimated using the time average over the signal's envelope, weighted by the signals energy.

#### Zero-Crossings

The zero-crossing count is the number of waveform sign changes in a given frame. This is computed after subtracting the DC offset (average amplitude) within each frame from the signal, and can be expressed as a zero-crossing rate per unit of time for each frame by normalising the count by the frame length [32].

### Spectral features

Spectral features characterise the distribution of frequencies across the magnitude spectrum for a given sound, within each STFT frame. The spectrum can be skewed towards higher frequencies, which is perceived as a brighter sound, or conversely towards lower frequencies, which corresponds to darker, muted sounds. Furthermore, the distribution of energy across the spectrum can either be concentrated in isolated peaks for tonal sounds, or have a broadband spread, which is perceived as a noisy, breath-like sound [32].

#### Spectral envelope

Analogous to the temporal envelope in the frequency domain, the spectral envelope corresponds to the overall shape of the spectrum, and can be computed by smoothing the energy spectrum of the signal. As stated in [41], the spectral envelope can characterise a sound independently of pitch, and therefore its shape is indicative of timbre. The following features seek to express this information more succinctly using a set of spectral metrics.

Spectral moments[32]

* The ***Spectral Centroid*** characterises the “central” frequency around which the signal’s energy is concentrated. It is calculated by the magnitude-weighted mean of the spectrum along the frequency axis. This can be interpreted as a broad measure of perceived “brightness” of the sound, in that it quantifies the proportion of high to low frequency energy [33]. But this does not account for the spread of frequencies; therefore this measure of brightness is especially indicative if the signal is distributed within a narrow-band of frequencies.
* ***Spectral Spread*** characterises how broadly or narrowly energy is distributed about the spectral centroid (the mean). It is measured as the standard deviation of the frequency distribution in the spectrum (weighted by the normalised magnitude of each bin). This measure is also equivalently described as the bandwidth relative to the centroid, for instance in [3].
* ***Spectral Skewness*** describes the skew, or asymmetry, of the spectrum about the spectral centroid. Negative values indicate energy concentrated below the centroid frequency, while positive values indicate the energy is concentrated in higher frequencies relative to the centroid.
* ***Spectral Kurtosis*** measures the spectrum’s flatness around the centroid. Particular ranges of the kurtosis value indicate different spectral shapes, as detailed in [32]: “[a kurtosis value of] 3 indicates a normal (Gaussian) distribution, < 3 a flatter distribution, and > 3 a peakier distribution". This allows us to describe with a single value the “peakiness” of the sound, which is an important part of characterising how tonal it is.

Spectral slope *(spectral tilt)*

The spectral slope is the gradient of the spectrum, typically computed using a linear regression over the points in the spectrum to find the slope of the spectral magnitude [32] or the log-power spectrum, depending on the definition used. This is another descriptor which, similarly to the spectral centroid and skewness, characterises the overall relative prevalence of high and low frequencies in terms of spectral energy.

#### Spectral Roll-off frequency

The spectral roll-off attempts to measure the cut-off point of the spectrum, as another descriptor of the spectrum’s overall shape. This is computed as the frequency below which a majority of the energy in the spectrum is condensed [33], for instance in [32], "the frequency fc(tm) below which 95% of the signal energy is contained" is used. This is particularly relevant in characterising low-pass signals, as the roll-off frequency will yield an estimate of the cut-off or corner frequency of a filtered signal.

#### Spectral Flatness Measure (SFM)

The SFM aims to measure how close the spectrum approaches white noise, whose spectrum is ideally flat. This is estimated by taking the ratio of the geometric mean to the arithmetic mean of the spectral amplitudes in a given frame [32]. Beyond describing the shape of the spectrum, flatness measures such as SFM and Spectral Kurtosis allow us to place the periodicity of a sound along a scale between tonal and noisy sounds, where on one end we have an ideal single sine tone, and on the other extreme white noise, which can be approached using an infinite sum of sinewaves of different frequencies uniformly distributed across the spectrum. The space between these extremes is occupied by sounds of increasing complexity as more tones are combined; this will be further explored in our discussion of harmonic features, in Section 3.1.5

### Spectro-temporal features

#### Spectrogram

The magnitude spectrogram of a signal is a spectral representation of the signal over time, made up of the magnitude spectrum computed over consecutive time frames. The resulting 2-dimensional matrix is typically plotted with frequency along the y-axis, and time on the x axis in visualisations using a colour intensity scale to show the magnitude of each time-frequency bin. This image is characteristic of the distribution over time of the input signal's energy across different frequencies. For instance, the fundamental and harmonic frequencies (see Section 3.1.5) and their associated intensities can be observed in this representation, as well as the evolution of individual frequency components over the signal envelope. Therefore the spectrogram gives a fairly complete representation of a signal's timbral profile, although it is not inherently pitch-invariant since pitch is linked to the frequency-axis.

Computation the spectrogram involves applying the STFT (see ) over consecutive frames in order to obtain the spectrum over time. For this, we must consider the effect of the frame length, , on the precision in both time and frequency of the spectrogram. A longer analysis window allows us to perform the STFT over more samples, which results in a higher frequency resolution since the resulting spectrum will contain more points. However, if the window is larger, the time resolution of the analysis decreases, since we calculate the spectrum over longer durations. If the window is too large, we risk no longer capturing any rapid changes in the signal over short durations, while if it is too narrow, we lose precision in the spectrum and risk losing harmonic detail in the spectrogram. This time-frequency precision trade-off, determined by the selected duration of the analysis window, is illustrated in [36], Fig. 4.2.. This trade-off can be somewhat moderated by zero-padding the analysis frame such that the window over which the STFT is performed contains more samples, resulting in synthetically higher frequency resolution using the same number of points in the original waveform. Paired with overlapping consecutive windows, this frequency-domain interpolation allows us to increase both the frequency and time resolution of the spectrogram, which ensures that the spectrogram captures the continuously time-varying information in a signal [35].

*TODO: Give example plots using different frequency and time resolutions to show the impact of both on the aspect of the spectrogram, and what shows timbral qualities best.*

In an implementation, one must therefore select values for the following parameters: the STFT frame length, the length of the analysis window (i.e. how much padding to be applied to the STFT frame), the spacing of consecutive windows (determining the amount of overlap between them), the window function used, as well as the frequency range considered. In selecting these, we must ensure that the degree of overlap between frames is not such that adjacent frames contain redundant information, and that the zero-padding is limited so as to not dominate the analysis window (which would cause inaccurate interpolation and artifacts).

To summarise, the computation of the spectrogram involves the steps shown in ***Figure 3.1‑1*** First, we sample equally-spaced frames of length and add padding, commonly by appending and prepending a sequence of zeros either side of each frame. Then we apply the STFT over each padded frame as described in , and take the power spectrum in order to represent the power per frequency and per frame in the spectrogram, discarding the phase spectrum.

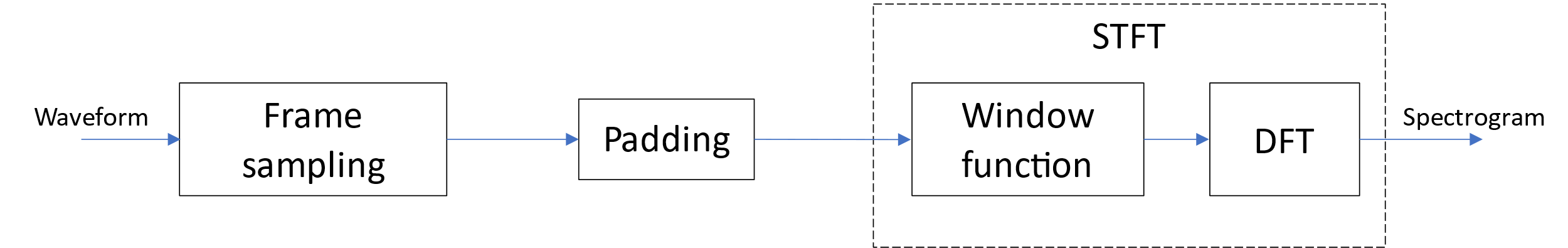


Figure 3.1‑1: Signal flow diagram summarising the steps involved in computing the spectrogram from a waveform. The output is typically converted to the power spectrum by taking the squared magnitude of the STFT result.

*Indicate which selections may be more appropriate to timbral analysis, as opposed to pitch or other analyses.*

#### Spectro-temporal envelope

The spectro-temporal envelope characterises the shape of the signal in both the frequency and the time domain, by encapsulating the evolution of the energy contained in each frequency bin over consecutive frames. The result is the shape of the signal over a given time period as a function of both time and frequency, and as stated previously, this can be seen in the spectrogram.

In [33], a feature characterising the spectro-temporal envelope is estimated for each frequency bin, calculated over a window of several frames by taking an average of the magnitude of each spectral component over consecutive frames (which the authors name “global spectral envelope”). The authors use this to derive “Harmonic Spectral Deviation”, which is a measure how much each spectral amplitude component differs from its neighbouring spectral envelope.

Spectral flux*(spectral variation)*

The spectral flux or variation is a measure of the spectrum’s rate of change over time. Two different definitions exist in the literature, both consisting of comparing the spectral distribution at consecutive time frames. Both calculations produce a function of time from the spectra of two successive time frames.

In [32], spectral variation is computed as one minus the correlation between two consecutive spectral amplitudes (normalised by the spectral energy at both time steps). In [33], the sum (over the spectral components) of squared differences between spectral magnitudes at two consecutive time frames is used.

### Harmonic features

In the context of harmonic analysis, complex sounds such as those produced by musical instruments are modelled as a sum sinusoidal components of differing frequency and amplitude, which are called partials. Of these, harmonic components of the sound are those located at integer multiples of the fundamental frequency (the pitch of the musical sound).

The fundamental is not directly implicated in timbral analysis, since timbre is defined as being independent of pitch, but this feature can be used for estimation of the expected harmonic frequencies of a signal as described in detail in part II.B.4. of [32], and potentially to normalise pitch-dependent features such as the spectrogram (see Section 3.1.4). Notably, the harmonic peaks present in a musical sound may deviate from the theoretical evenly-spaced harmonic distribution along the frequency axis, which the authors of [32] qualify as “inharmonic distortion”. The harmonic modelling context introduces a number of features which are commonly used to analyse the timbre of musical sources by characterising the distribution of harmonics (harmonic centroid, spread and variation) as well as the extent to which the signal conforms to harmonic assumptions (i.e. the tonality of the sound).

Harmonic Centroid [33]

The harmonic centroid frequency, analogous to the first spectral moment, is the amplitude-weighted mean frequency of the harmonic peaks identified in the spectrum.

Harmonic Spread [33]

The harmonic spread, similarly to the second spectral moment, is measured as the amplitude-weighted mean across the detected harmonics of the standard deviation of each of the harmonic peaks. This is expressed normalised by the harmonic centroid frequency.

Inharmonicity and Harmonic energy skewness[3]

Inharmonicity is defined in[3]as the measure of how much the first 4 partials differ from the corresponding theoretical harmonic frequencies (integer multiples of the fundamental). This is computed as the sum of the distances on the frequency axis between each partial and the corresponding expected harmonic frequency (each distance is normalised by that harmonic frequency). The calculation of harmonic energy skewness is similar to that of inharmonicity, but each distance is scaled by the spectral energy in a neighbourhood of the considered partial, yielding a measure analogous to the third spectral moment (skewness).

Harmonic Variation [33]

The harmonic variation is similar to the spectral flux between consecutive time frames. It is computed by the normalised correlation between the harmonic peak amplitudes between two consecutive frames.

#### Harmonicity Rate and Harmonic Brightness

These features, as defined in [33], both aim to characterise the prevalence of harmonic content in the sound, giving a measure of how tonal or noise-like (inharmonic) the sound is.

The harmonicity rate aims to measure the proportion of the wider magnitude spectrum that corresponds to the harmonics of the sound. This is estimated by computing the maximum of the normalised autocorrelation of the signal. The harmonic brightness aims to quantify the prevalence of upper harmonics (higher frequency overtones) in the signal by taking the ratio of the sum of index-weighted partials’ magnitudes over the sum of the harmonics’ magnitudes.

#### Tristimulus values

Developed as an analogy to the three channels of visual colour, these coefficients aim to characterise the harmonic “colour” of a sound based on the distribution of energy over the harmonic series in the signal. Their computation, detailed in[32], relies on the sums of the amplitudes of the harmonic partials of the signal in a given frame.

#### Odd-to-Even harmonic energy ratio

Sounds containing mostly even harmonics are perceived as "smoother" than those in which the odd harmonics dominate the share of energy in the spectrum[32]. The odd-to-even ratio is computed by the sum of squares of the odd harmonic amplitudes divided by that of the even harmonics.

### Formant analysis and the source-filter model

Formant analysis is a popular approach in speech timbre analysis and synthesis [42], often applied to speaker differentiation and identification [43], and can be compared to harmonic analysis in the musical context (formants in speech processing corresponding to harmonics in musical contexts). The approaches and features involved in formant analysis and source-filter modelling could provide useful results in characterising the timbre of musical instruments via analogy to the human voice.

#### Source-filter model and formants

The source-filter model, typically applied to modelling the human vocal tract, interprets a sound as resulting from a linear system, by which a source (exciter) being passed through a filter (resonator), as shown in **Figure 3.1‑2**. The excitation at the model source accounts for the noise-like qualities, while the order and characteristics of the filter account for the resonant (tonal) qualities of the resulting sound.

Formant analysis concerns the study of the resonator, and consists of determining the resonant frequencies and bandwidths of the filter modelled for the signal, which are particular to the shape and nature of the body generating the sound. As discussed in [44], the relationship between these formant frequencies is relatively constant across different pitches played by musical instruments, indicating that the formant frequencies, magnitudes and bandwidths are a relevant set of features for characterising instrument timbre. Furthermore, many of the computations described in Section 3.1.5 could be applied to characterising formants and their prominence in the spectrum analogously to harmonics.

#### Linear Predictive Coding (LPC)

The most common scheme for estimating the frequency and magnitude of formants from a waveform is Linear Predictive Coding, which predicts each value of the signal by linear combination of previous samples, as described in [42]. This corresponds to an auto-regressive filter model, which is computed by using the least squares solution to determine each filter coefficient (linear prediction coefficients) using a pre-determined order for the filter. The resulting LPC filter is an estimate of the filter part of the source-filter model, and the location of the filter’s poles in the z-plane yields the formant frequencies (the peaks in the filter’s frequency response), as shown in **Figure 3.1‑2**.

Machine generated alternative text:
s[n] E akS[n — k] + e[n] 
Pulse/noise 
excitation 
Vocal tract 
e[nl H(z) = 1/A(z) 
s[nl 
z-plane 

Figure 3.1‑2: Auto-regressive filter equation with coefficients ak, error e[n] and order p (top).   
Source-filter model showing z-domain filter transfer function H(z) and representing an illustrative frequency response and pole locations in the z-plane (bottom). Source: From [45], slide 7

*TODO: detail initial LPC formant analysis experiment, with plots*: Formant extraction and analysis using Linear Predictive Coding coefficients. Wrote a MATLAB script to extract formants from single-note recordings of a flute played at different pitches. LPC functions provided by the VOICEBOX toolkit were used to estimate LPC coefficients from the waveforms, which were then translated to estimated formant frequencies and bandwidths. The relative frequencies of the first few formants were plotted across the different pitches in the range of the instrument in order to confirm the pitch-invariance of the ratio between the formant frequencies of a given instrument, which is one of the reasons for which formants are considered as descriptors of timbre.

#### Inverse filtering

Inverse filtering complements formant analysis by attempting to model the excitation, or source, part of the source-filter model through applying an inverse filter to the signal in order to recover the output of the “source” element. The excitation can be estimated by applying Linear Predictive Coding and considering the linear prediction error term e[n] shown in **Figure 3.1‑2**, which is called the residual [46].

This atonal part of the sound corresponds to unvoiced sounds in speech, and analogously concerns the breathy, inharmonic aspects of the sound produced by musical instruments. Thus, isolating and characterising this excitation may provide interesting results in the way of encapsulating timbre beyond harmonic information.

### Cepstral features and the Mel scale

The Cepstrum of a signal is obtained by taking the discrete cosine transform (DCT) of the log-magnitude frequency spectrum. This Cepstral representation shows peaks corresponding to shifted echoes in the original waveform, and therefore reveals a representation of periodic events in a waveform, such as the periodicity corresponding to the fundamental pitch and formants in a complex signal such as speech [47].

#### Mel-Frequency Cepstrum Coefficients (MFCCs)

The Mel scale is a logarithmic frequency scale based on human perception of pitch relationships, relying on the fact that the ear can more finely differentiate between equally spaced low frequency sounds than their higher frequency counterparts. The Mel-frequency Cepstrum is obtained by mapping a signal’s spectrum to the Mel scale and computing the DCT of the logs of the spectral magnitudes across the Mel scale [47]. This Mel Cepstrum has discrete values which form the Mel-Frequency Spectrum Coefficients, a feature set which encapsulates pitch and harmonic information emulating human perception. These coefficients are therefore compact and powerful descriptors of the perceived harmonic content of a signal over time.

#### The Mel scale, perceptual features and the log-Mel spectrogram

The relationship between frequencies on the Hz scale (which we refer to as DFT or STFT frequency) and their mappings on the Mel scale is given by the following equation (adapted from [48], slide 91):

This mapping is plotted (in the discretised form produced by a Mel filter bank) in **Figure 3.1‑3**, where its approximately linear shape at low frequencies, and logarithmic shape elsewhere, can be observed.

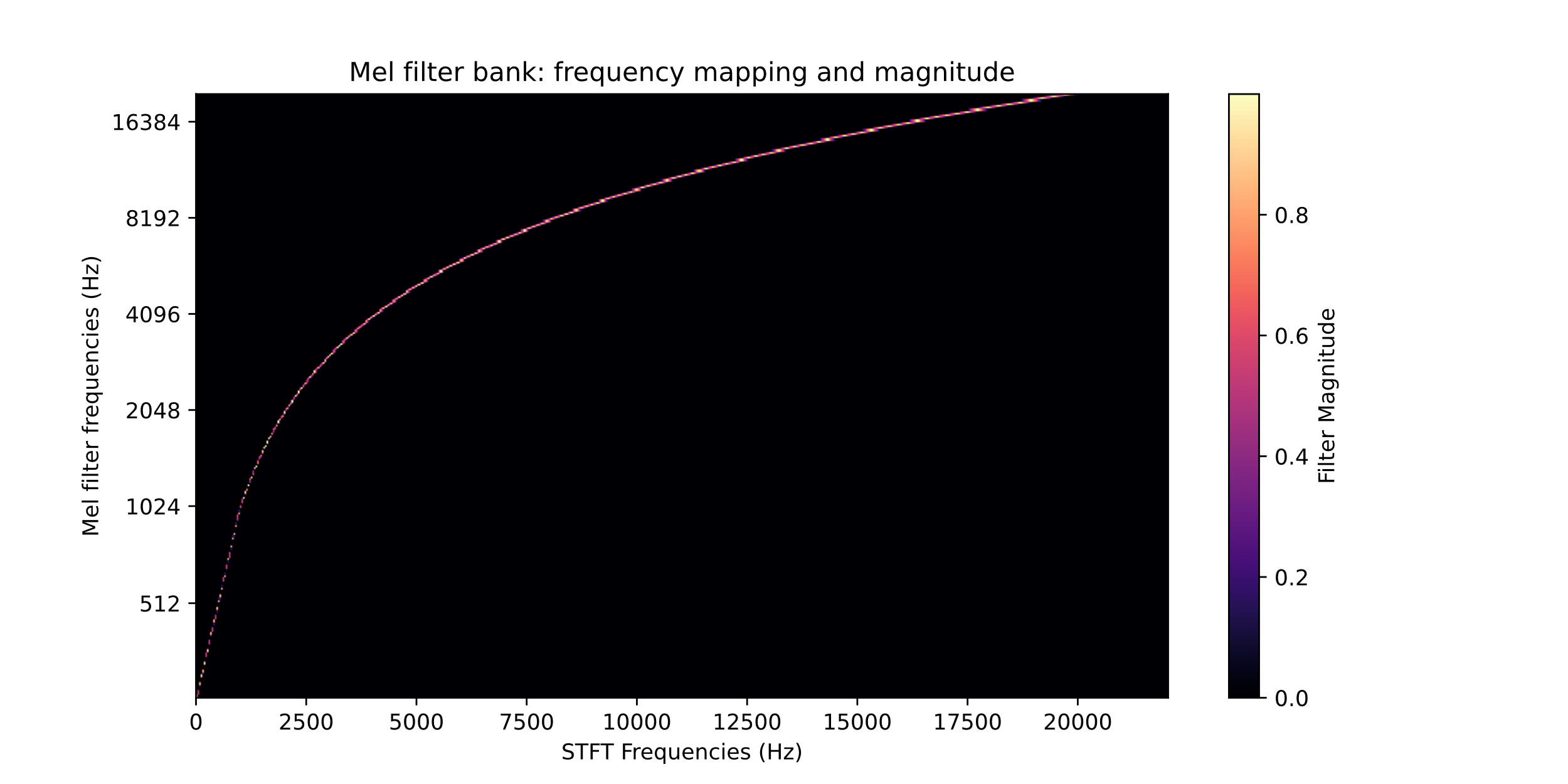


Figure 3.1‑3: Plot showing the mapping from STFT frequency to Mel-scale frequency, in the case of a 300-filter Mel bank (whose frequency response is shown in Figure 3.1‑4).   
This plot shows how the triangular filters in the Mel filter bank approximate the continuous logarithmic Mel scale, such that the mapping is applied with a uniform unit magnitude across all the considered frequencies.

In practice, in order to apply the Mel scale to a signal (for computing the MFCCs, for instance), a Mel filter bank is constructed and applied to the signal. The Mel filter bank, also called Mel basis, is a set of triangular overlapping filters whose bandwidths are spaced according to the Mel scale [48]. **Figure 3.1‑4** shows the aspect of a Mel filter bank’s magnitude response over different frequency regions. Therefore, filtering a signal with this bank approximates the transformation of the spectrum’s frequency axis to the Mel scale, and can be applied by convolution in the time domain or multiplication in the frequency domain.

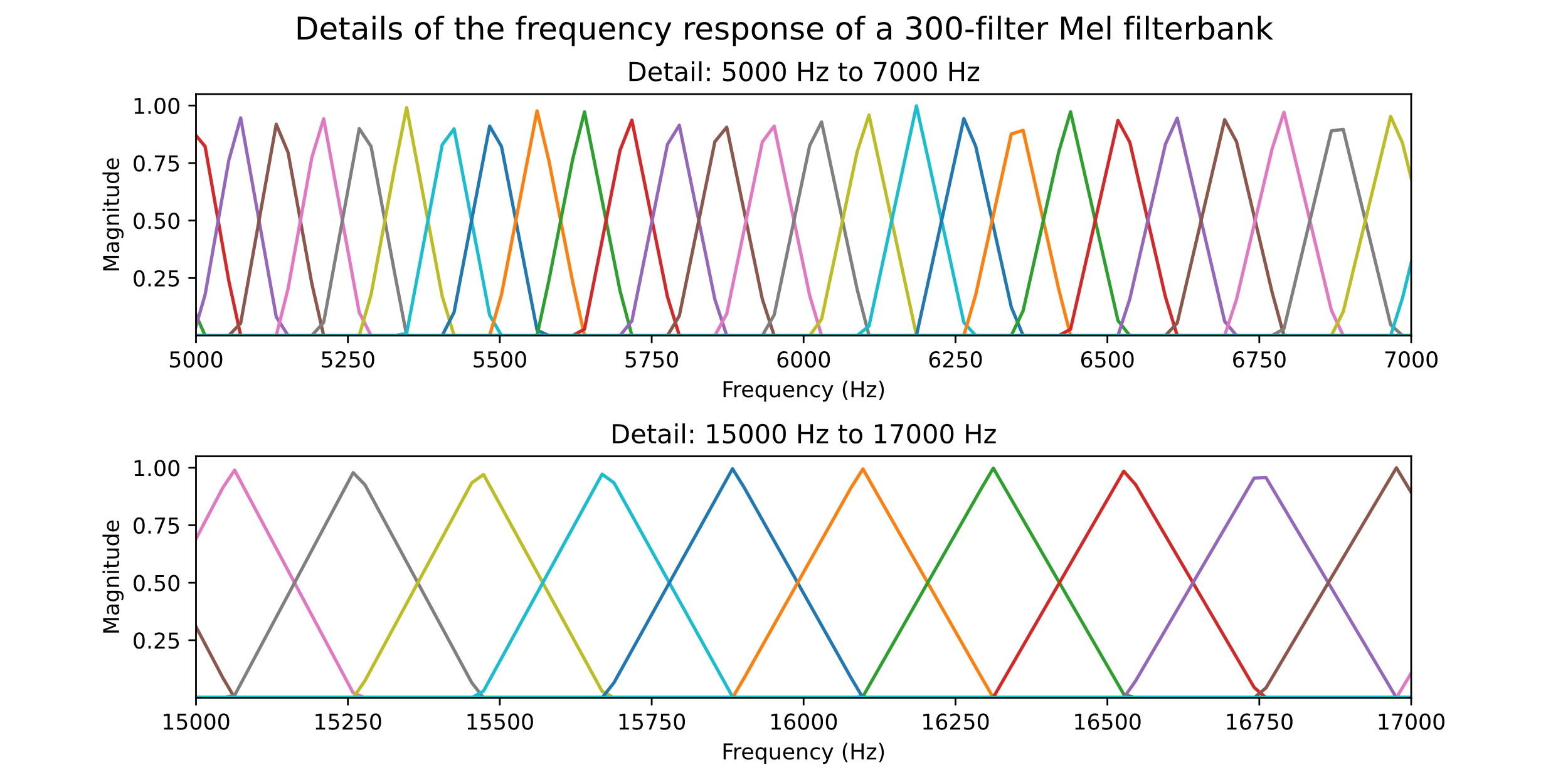


Figure 3.1‑4: Mel filter bank magnitude response plots corresponding to a 300-filter Mel bank.   
Each coloured line is a separate filter’s frequency response, each having a triangular shape and roughly unit gain. The top figure shows a lower-frequency detail of 2 kHz bandwidth within the full magnitude response of the filter bank, while the bottom figure shows a detail of the same bandwidth in a higher-frequency portion of the spectrum. Note the higher density of filters in the top plot in order to achieve higher Mel-scale resolution in the lower frequencies.

An important parameter involved in the application of the Mel scale is the number of filters to be used in the construction of the Mel bank. This essentially controls the frequency resolution of the Mel scale discretisation, but the resolution of the resulting Mel spectrum also depends on the number of points used in the DFT computation (which determines the spectrum’s native frequency resolution). Therefore the number of Mel filters can be selected so as to maximise the resolution of the Mel spectrum, while keeping in mind that too large a number relative to the DFT’s resolution will result in empty frequency bins being produced, and will increase the computational complexity of applying the scale.

The Mel scale biases the resolution of the frequency axis to scale with the ear’s ability to differentiate between pitches, making it appropriate for application to musical acoustic analysis. For example, this can be observed empirically by the roughly logarithmic distribution of note pitches on the piano keyboard with respect to their fundamental frequency, as illustrated in **Figure 3.1‑5**, a standard which, like the Mel scale, reflects human perception of acoustic frequency.

Figure 3.1‑5: Plot presenting the relationship between the 88 notes on the piano keyboard and their fundamental frequencies.   
We note here how this relationship closely resembles that between DFT frequency and Mel frequency plotted in Figure 3.1‑3, supporting the idea that the Mel scale is adapted to musical-domain analysis. Piano key fundamental frequencies obtained from [49].

Taking the log of the magnitudes, or similarly applying the decibel scale to the power spectrogram similarly mimics human perception, as this emulates the human ear’s logarithmic response to acoustic amplitude. Thus, both the frequency axis and magnitude range in the spectrogram can be warped to approximate human perception of pitch and volume respectively. A representation which applies both of these perceptual characteristics is the log-Mel spectrogram, which uses the Mel frequency scale for the y-axis and a logarithmic magnitude scale for the intensities in each bin. The log-Mel spectrogram is therefore a powerful standalone feature encapsulating spectro-temporal information by leveraging perceptual features, an approach which is applied for instance in [50] as detailed in Section 2.2.2. To summarise the computation of the log-Mel spectrogram, we illustrate the steps involved in **Figure 3.1‑6**.

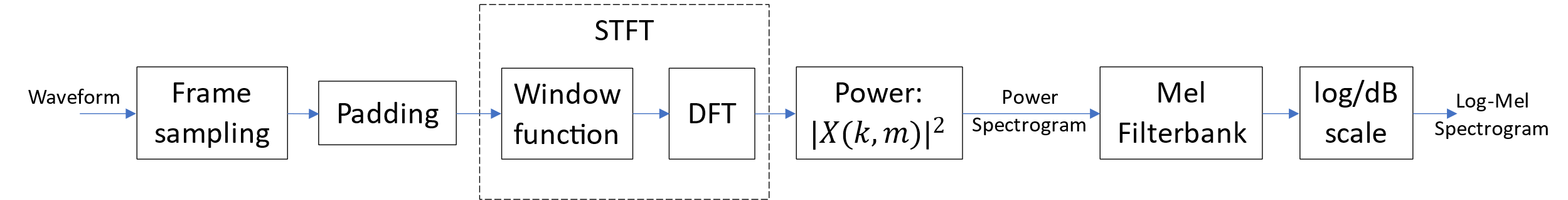


Figure 3.1‑6: Signal flow diagram summarising the steps involved in computing the Log-Mel spectrogram, including computation of the power spectrogram (see Figure 3.1‑1) and application of the Mel and log or decibel scales. Adapted from [48], slide 92.

## Machine learning applied to timbral identification

A popular application of timbral analysis of musical audio signals is the automatic classification or grouping of musical instruments from audio recordings; we will guide our attention to the various ways in which musical instruments can be identified from their timbre. In order to draw a classification label from timbral input features, or to cluster samples corresponding to instruments with similar timbres based on an input feature set, many inference algorithms have been applied in the literature. This section gives an overview of the machine learning classification and clustering methods most commonly applied to timbral analysis tasks, especially in the context of musical instrument identification by timbre. We will discuss the most popular methods for timbral classification, while also mentioning clustering methods which are also used in the context of timbral analysis; and then explain the motivation behind our focus on neural network classification.

### Non-neural network methods

Traditional machine learning algorithms generally require careful pre-processing of input waveforms into low-dimensional input features to infer the timbre qualities of a signal. These inference models therefore inherently require more structured data as input in order to make informed decisions for classification and clustering, but some are advantaged by their simplicity and their ability to make decision using a smaller amount of data, for instance for methods that do not require training, as opposed to neural network models.

#### Timbral classification (supervised inference)

A supervised classification system seeks to assign one or multiple labels from a pre-defined set to an unseen sample (from the test set), based on the seen samples’ known labels (sometimes called the training set if the learning process involves training). This inference results from relating the set of input feature values taken by the unseen example to the set of input features associated with the known labels.

K-nearest-neighbours (KNN) is a simple yet popular scheme for classification of unseen samples given a set of known examples, as described in [11]. The method consists of plotting all samples on the input feature space, such that each known sample forms a point of known label and coordinates in the space resulting from the values taken on by the features for that sample. When presented with an unlabelled sample, the KNN algorithm simply classifies it using the most prevalent label out of the K nearest samples (usually measured by Euclidean distance in the space), where K is a parameter of the method. Variations on this method include distance-weighted voting from the K neighbours. This algorithm performs poorly, however, when using a greater number of input features, as the feature space becomes increasingly sparse according to the “curse of dimensionality” (a concept coined by Richard Bellman and detailed in [51]). Additionally, its ability to generalise inference to unseen data is limited due to the local nature of the connections established between data points, as noted in [3].

Decision trees (or binary trees) is another straightforward classification scheme described in [11] which builds a tree structure based on the distribution of each feature’s values across a training set. This structure divides the known samples such that each branch groups all the samples taking a particular common range or class for a given feature, by splitting the tree on the point subdividing the feature space with maximum information gain (i.e. entropy reduction). When an unseen sample is input, the tree is traversed from the top down, at each node taking the branch corresponding to the test sample’s input feature value, until a leaf node (where all seen samples grouped by a branch share the same label) is reached and its label is assigned.

Other classification algorithms often cited in timbral analysis work include Support Vector Machines (SVM) and Discriminant Analysis (e.g. Linear, Quadratic, Canonical) as detailed in [3], though these schemes have in recent years fallen out of favour in machine learning research due to their implementation complexity, instead replaced largely by neural network classification.

Despite their limitations, simply-implementable classification schemes such as KNN and decision trees remain useful tools to pre-validate the quality of a choice of input features or data when exploring a classification problem, before moving on to developing a more generalisable classifier such as a neural network model. In particular, the maximum information gain strategy in the construction of a decision tree could help elucidate the most telling features in discriminating between instrument timbres, as noted in [11].

#### Timbral clustering (unsupervised inference)

Clustering methods differ from classification schemes in that known examples with output labels are usually not supplied; the models are left to relate (or group) samples in a set with one another without supervision, only using their input feature values. This provides a viable alternative to classification for inference when manual ground truth annotations are not available or inconsistent. For timbral analysis, clustering methods such as Gaussian Mixture Models (GMMs) (described in [11]) and Self-Organising Maps (SOMs) are applied in the literature to plotting a low-dimensional timbral space for visualisation and quantification of the relationship between musical instrument sounds [9]; or to provide a system for indexing audio databases by timbral similarity [52].

*TODO: Keep the sections which don’t concern Neural Networks as studied background material, but mention why I didn’t decide to try to take these forward.*

### Neural Networks and Deep Learning methods

#### Neural Networks

In the context of more complex pattern recognition and feature extraction from data, many of the machine learning algorithms previously discussed have been superseded in recent years by gradient-based backpropagation learning of multilayer Neural Networks (NNs). As explained in detail in [10], this supervised learning method relies on samples being input to a network of nodes called a multi-layer perceptron, which models a learnt non-linear function. Each layer (set of nodes) of the network is connected to the next using a linear combination between the input values and the layer parameters, followed by a differentiable non-linear activation function. Each layer’s parameters is made up of a weight which multiplies the input for each pair of input-output nodes, and an added bias for each output node. The output value of node at a layer , denoted , given a set of inputs is defined in , adapted from [53].

where:

* is the output value at node in layer .
* is the vector of inputs to the considered layer.
* is the vector of learnable weights of the linear combination between the layer inputs and the considered layer output node , unique to node in layer .
* is a learnable scalar bias unique to node in layer .
* is a non-linear activation function applied to the result of the linear combination.

can therefore be used to compute the output value at each of the hidden or output layer nodes in ***Figure 3.2‑1*** in the forward computation, and the same applies to networks with more hidden layers, feeding values from the input of the network to its output.

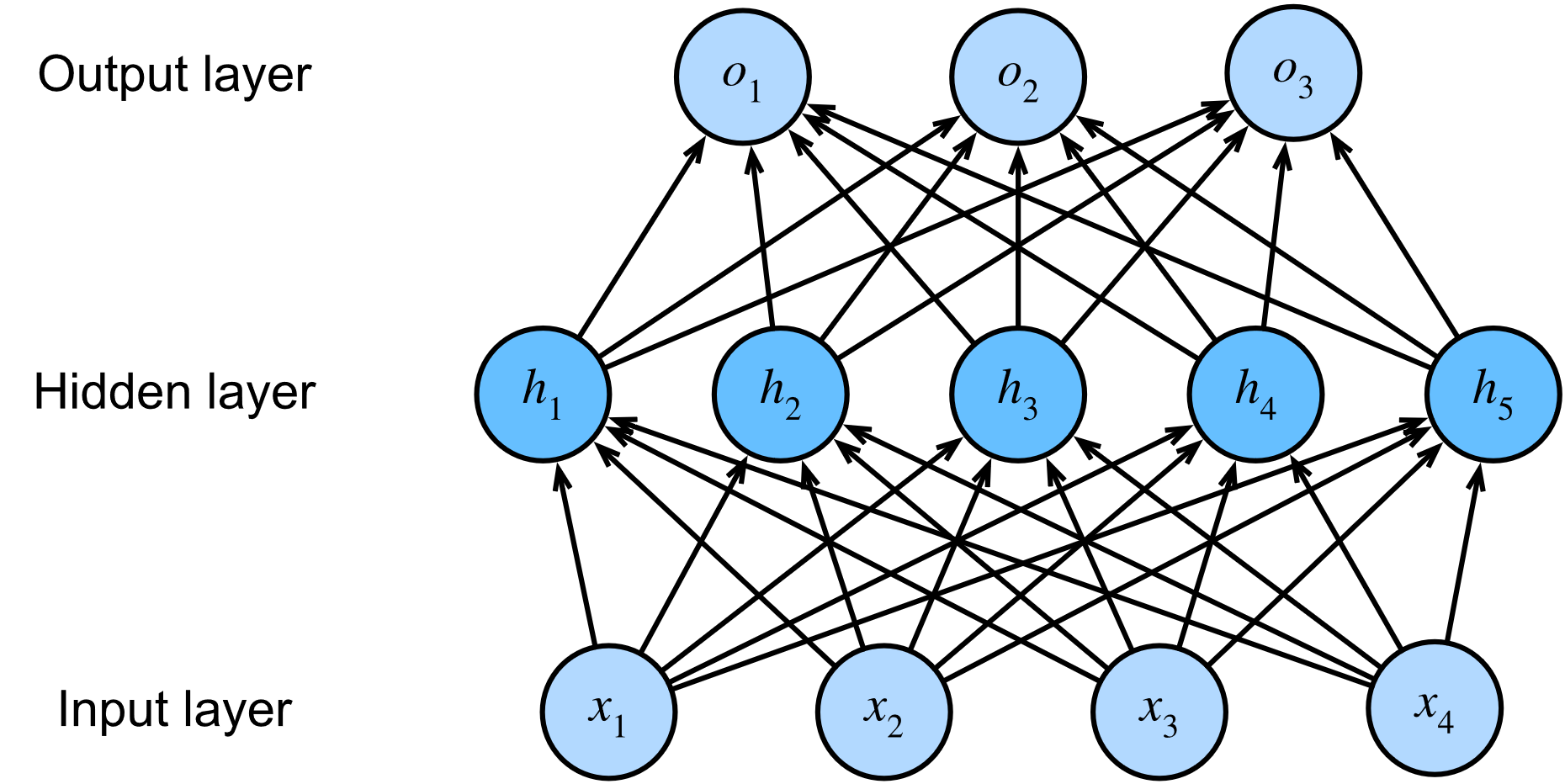


Figure 3.2‑1: Diagram of a 3-layer fully connected neural network, containing 4 input nodes, 1 hidden layer with 5 nodes, and 3 output nodes showing arrows between the nodes of consecutive layers where linear combination weights are applied to connect nodes from one layer to the next. The non-linear activations are not shown, but are implied as being applied after the linear combination at each “hidden” or “output” node. Figure adapted from [53].

When training samples are input to the network, the weights are randomly initialised, and the value for each input feature is fed through the nodes via the weights and activations in each layer in a forward pass to produce the activation outputs of the final layer of nodes, which is compared with the ground truth label corresponding to the given known example. The resulting error, which is computed using a specified loss function, is then used to update each connection weight using a chosen optimisation algorithm. In Stochastic Gradient Descent (SGD) optimisation, the goal of the optimisation given a batch of inputs is to take a step towards minimising the error between the batch network outputs and batch labels, informed by the error gradient (from which the direction of descent towards a local minimum can be derived). The error gradient is backpropagated through the network using the chain derivative rule, so that the error gradient with respect to the parameters of each layer can be computed to inform the updated value for each weight in the network.

Classification problems are typically handled by setting the number of output layer nodes to match the number of possible classes, with an output activation function bounded between 0 and 1, and encoding a ground truth label during training with “1” for the output node corresponding to that class and “0” for all the others. For example, in the case of binary classification (discrimination between two classes), only a single output node is required in the final layer. Then, at test time, the input values of a given test sample are passed forward through the network, and the class corresponding to the output node with the highest activation is selected as the model’s prediction.

#### Convolutional Neural Networks (CNNs)

Relative to fully connected neural networks and other machine learning methods, Convolutional Neural Networks have been shown to bring notable performance improvements to perceptual inference tasks such as image classification, computer vision and, given the suitable pre-processing is applied (e.g. generation of spectrograms), audio processing. The same training process, nonlinearities, and hyperparameters detailed previously apply to CNNs; however the structure of CNNs differs greatly from that of NNs. From a high-level perspective, the key difference between traditional NNs and their convolutional counterparts is their handling of high-dimensional inputs, since NNs operate natively on 1-dimensional vectors while CNNs operate directly on maps of dimension 2 or above, without needing to flatten to an array. While standard NNs have all nodes in a given layer’s input individually connected to each node in the output of the layer with a unique weight, convolutional networks use a weight-sharing structure in which the layer connections are shared between multiple input nodes, or pixels in the image domain.

In 2 dimensions, convolution is achieved by applying the spatial filter kernel of size using a sliding window across all locations in the input map of size to produce an output map . The output of a convolutional layer at pixel location is given by the sum of products of the input values by the kernel weights within the considered window of size , centred around location , as defined in (adapted from [54], slide 7).

Where are commonly chosen to be odd numbers in order to centre the kernel’s sliding window on the output pixel’s location . Note that in the context of CNNs, as opposed to the signal processing definition of convolution, this operation does not require the filter kernel to be flipped before taking the entry-wise sum of products, since the filter weights are learnt by the CNN in either case, as explained in [55]. Border pixels in the input map may be ignored, or padded by the appropriate amount in order to produce an output map of the same shape (i.e. and) [56]. From we can see that the output pixel at location only depends on the learned filter weights and the input pixels in a small region around . In a convolutional network layer, to this value is added a learned bias, which is a single scalar applied across the whole map.

CNNs are made up of a series of these convolutional connections between layers as illustrated in, with each convolution followed by a nonlinear activation function as with standard NNs. In each convolutional layer, independent 2 dimensional learnable filters can be stacked along the 3rd dimension, producing corresponding channels in the output map, in order to extract multiple features in parallel paths and operate on -channel data such as a 3-channel RGB image. Thus each convolutional layer has learnable parameters, independently of the input/output map dimensions. This is generally much fewer than the equivalent fully connected NN layer, which would depend on the map dimensions, requiring parameters; therefore CNNs have fewer optimisable parameters between layers, reducing the cost of each optimisation step and the size of a model. In some variations, a combination of different sizes of filters can be used even within a single CNN layer, allowing analysis and transformations can be applied at multiple scales and levels of abstraction in a single model, for example as seen in [17].

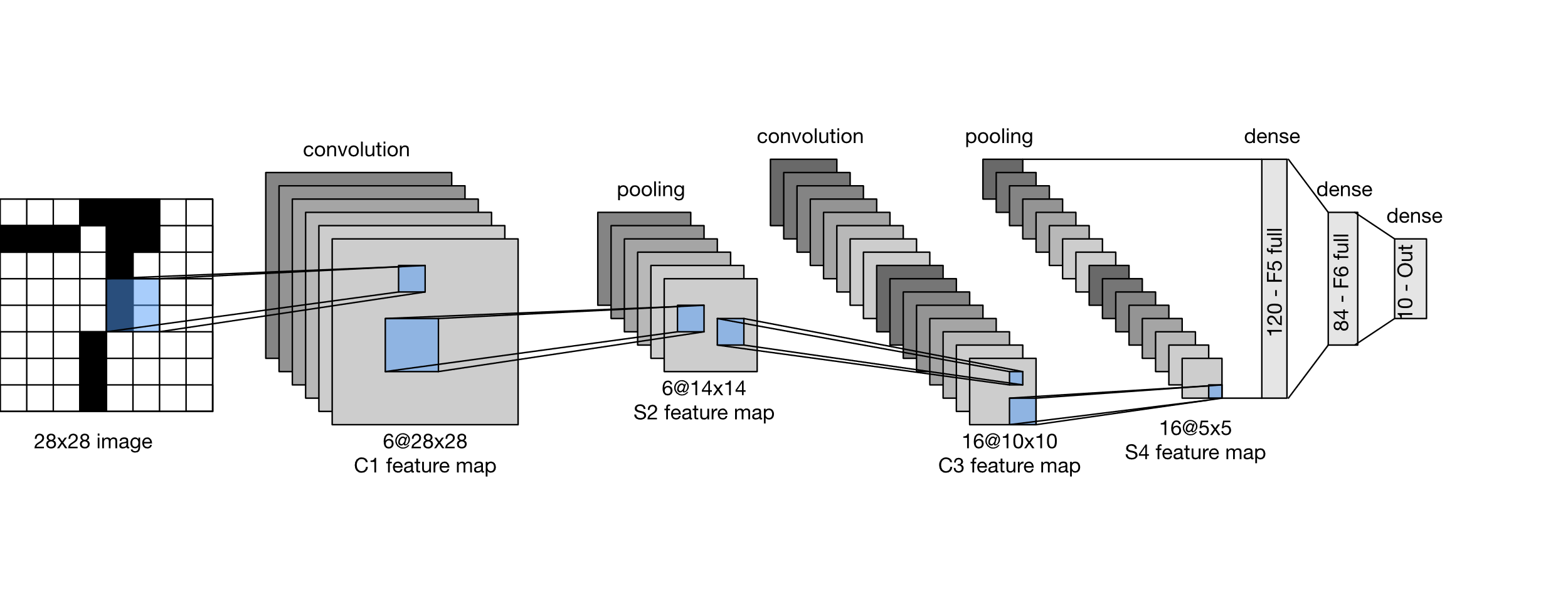


Figure 3.2‑2: Diagram of a simple convolutional neural network architecture, namely LeNet [12] which classifies 28x28 input images as belonging to one of 10 classes.   
The network contains 2 convolutional layers interspersed with 2 pooling layers. For instance, the first convolutional layer, C1, uses a 6-channels of 2x2 filters, producing 6 feature maps (labelled “C1 feature map”) which are the same shape as the input images since the input images are padded by 1. The output of the last pooling layer is flattened into an array to be processed by a small fully-connected neural network whose layers are labelled “dense”, in order to produce a 10-element array. Figure adapted from [57].

***Figure 3.2‑2*** shows the typical structure of a simple CNN, whose final layers are fully-connected “dense” layers. These dense layers can be replaced, as it is possible to continue reducing the dimensionality of the 2-dimensional feature maps using convolution operations so as to produce the correct number of output activations. Such architectures are called Fully Convolutional Neural Networks (FCNNs) and have the advantage of retaining spatial information at different levels of abstraction until the final output layer.

Dimensionality reduction between consecutive layers can be achieved by strided convolution, or by pooling (down-sampling by grouping regions) patches of the feature map by average or maximum value, as detailed in [58]. For these reasons, CNNs are readily adapted to dimensionality reduction and to fusion of high dimensional features, for instance in the context of fusing timbral features as in [59]. This dimensionality reduction achieved by series of convolutions and pooling operations allows the network to output a low-dimensional vector given a large-scale input image, in order to perform classification for instance. Another key advantage of CNNs demonstrated in [58] is that the combination of convolution and pooling operations yields approximate translation invariance, making the network less sensitive to input locality. This means that a series of CNN layers, each made up of convolution followed by pooling, produces a similar response to a given input pattern regardless of where the pattern appears in the input map. This is a desirable quality for image recognition applications in which a visual shape needs to be recognised regardless of its location, but also when applied to timbral analysis using spectrogram inputs, since it allows the network to make inferences independently of the timing (location on the x-axis of the spectrogram) and pitch (y-axis of the spectrogram) of the signal.

*TODO: Add a section on how to typically interpret CNNs: visualising filters/activations at different layers to observe different level of abstraction, inversion/inception*

#### Common training and data considerations for neural networks

In general, on top of choosing which type of neural network architecture to use for a classifier problem based on the nature of the given problem and input features, designing a neural network involves the choice of a large number of architectural parameters, including the number of nodes and activation functions in each layer, and the total number of layers in the network. In the case of CNNs, we choose the number of convolutional filters to apply between layers, as well as their dimensions.

A pitfall commonly encountered when using a Deep Neural Network (DNN) structure, i.e. a NN containing many sequential layers, is the issue of vanishing gradients. This refers to the problem of the gradients being applied during backpropagation being small in magnitude such that the applied optimisation updates are too slight, causing the learning process to stall before convergence. This occurs since backpropagation involves calculating the gradient of each functional component (linear layers and non-linear activations) in the network with respect to the optimisable parameters, and combining them by applying the chain rule. If activation functions in the chain are too flat in the region of differentiation, their gradient will be small. Because the chain rule is multiplicative, when combining the gradients from the output layer backwards through each layer in the network to reach a given layer, each small gradient in this path will shrink the magnitude of the parameter update applied. Deeper networks are thus more affected, as the update applied to the earlier layers can shrink exponentially as we add more layers to the end of the network.

The vanishing gradient effect can be mitigated by selecting appropriate activation functions for the network, highlighting the importance of the selection of non-linearities to use in the NN. For instance, Sigmoid [53] and other smooth functions such as Tanh are subject to vanishing gradients, while ReLU [53] and its variants are rectilinear so as to prevent small gradients, as illustrated in ***Figure 3.2‑3***. Therefore, we will generally prefer using ReLU-like activations in the hidden layers of deeper neural networks.

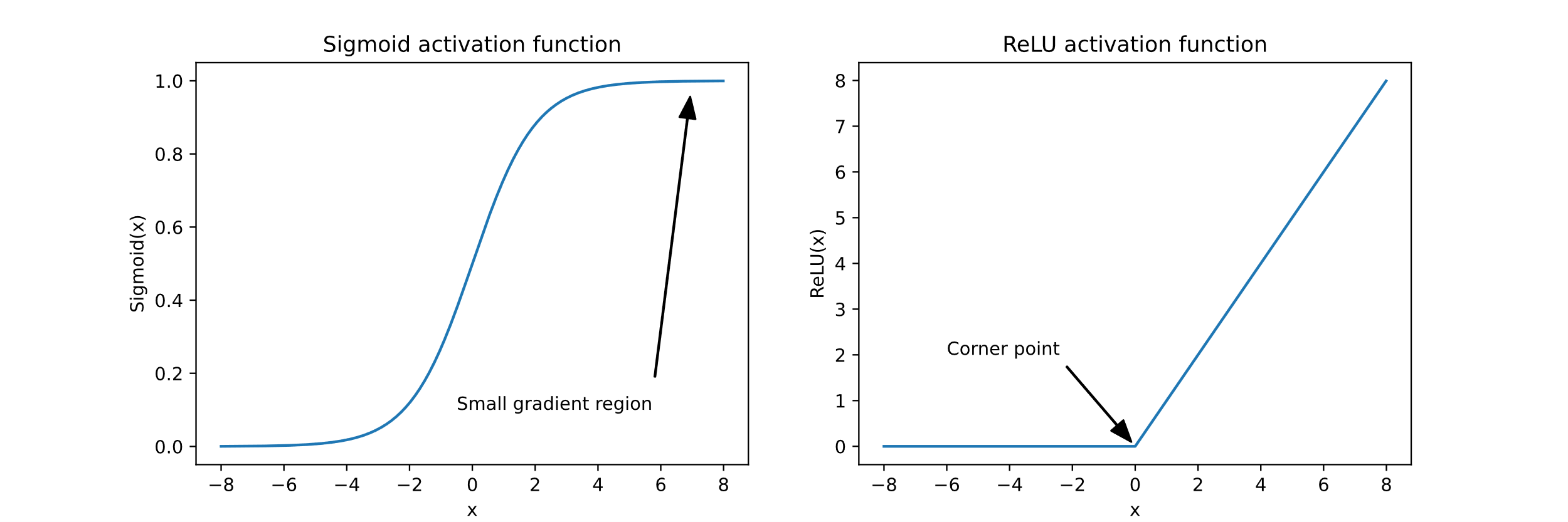


Figure 3.2‑3: Comparison of a smooth activation function (Sigmoid, left plot) subject to vanishing gradients to a linear-by-parts ReLU activation function (right plot), whose gradient is either 0 or 1 depending on the sign of the input, and is therefore not subject to vanishing gradients. The formulae for both of these functions are given in and .

Added to these architectural choices are training hyperparameters, such as the number of times random batches of training samples are passed through the network (the number of training epochs), the size of each batch of training samples (the batch size), as well as the optimisation algorithm and its parameters including the optimisation step size (the learning rate). Finally, a loss function tailored to the tackled problem must be selected so as to model the type of error we seek to minimise by training the NN. While a sum of squares error is usually adapted to regression problems, classification error is usually measured using loss functions that measure the mutual information between the ground truth and the network outputs, such as cross-entropy. The formula for the cross-entropy loss, or log-likelihood loss, between a network output prediction and the ground truth label of a -class classification problem is given by (adapted from [60]).

In the case of a binary classification problem in which , this function is called binary cross-entropy loss. Note that in the minibatch Stochastic Gradient Descent optimisation scheme typically applied to neural network training, optimisation is performed over a batch of data drawn from the training set, as detailed in [61]. Therefore, the sample mean of the loss function is taken over a batch of labels and predictions, in order to compute the estimated error of the current model, and this is then applied in one step of backpropagation.

Deeper neural networks require a large amount of training data in order to learn complex patterns [17]. Therefore, a number of considerations are required to properly handle data so as to allow a generalisable model to be learnt, especially to prevent overfitting a model to a particular dataset and to avoid bias. As a result, when designing a Neural Network classifier, and in general for supervised machine learning systems, we must split the dataset in order to train, select and evaluate models. This process, as detailed in [62], requires partitioning the data into three non-overlapping subsets: one subset for training, one for validation and one for testing. A training subset is required to expose the classifier to a set of inputs paired with the corresponding annotations in order for it to learn to predict the outputs through training – this subset typically contains the largest portion of the full dataset so as to provide a wide range of examples to help the model generalise. The validation subset is used to evaluate trained models during development and to observe the effects of tuning network architecture and parameters on its performance on data not seen in the training process. However, if fixed training and validation sets are used throughout the development process, we are likely to bias the models towards overfitting to the validation set; therefore, we require a third, truly unseen data subset. The held-out test set is thus used at the end of development to gauge the quality of the final model’s predictions on unseen data.

Further to careful data partitioning, strategies to mitigate overfitting in Neural Network classifier design include ensuring the employed dataset is class-balanced or representative of a wider population, data normalisation to ensure each input feature is considered equivalent by the network at the start of training, data augmentation, transfer learning [63], and regularisation measures [64]. Regularisation, in general, aims to penalise or prevent over-specified high-variance models during the training process, since these are likely to overfit the training data. This can be achieved by simply injecting random noise into the training data, or by adding a penalisation term to the loss function in order to constrain the magnitude of the model parameters (e.g. L1 and L2 regularisation). Another form of regularisation is batch normalisation, which, as further detailed in [65] adds learnable layers which track the batch means and variances during training, and uses them to standardise the previous layer’s output before feeding it to the next layer. Similarly to dataset normalisation, this method has been found to improve training performance considerably, a benefit which is believed to result from the injection of further noise given the use of batch sample statistics.

#### Other relevant Neural Network variants

Recurrent Neural Networks (RNNs), and in particular Long Short-Term Memory (LSTM) networks, are designed to exploit sequential data such that a dynamic response to time-sequential inputs, such as audio, can be learnt by taking as input consecutive frames of features instead of a single time-sample at a time. The modelling of dependencies within a sequence of inputs is achieved using memory states, which allow information to be retained from one input step to the next. These recurrent connections are trained using “backpropagation through time”, in which the error gradient resulting from the output at a particular time step is passed to elements in the network processing the other time-steps in the sequence [66]. Since they are designed to model the temporal relationships between consecutive inputs, these models are very commonly applied in the fields of speech recognition, Natural Language Processing (NLP) [17], and MIR tasks [24]. Therefore, these architectures are well-adapted to analysing the timbre of musical audio by explicitly capturing its temporal qualities, and could provide improvements over systems that do not take into account temporal dependencies on a structural level.

As network architectures have increased in complexity as enabled by larger datasets and more capable hardware, a recent development is end-to-end learning. This powerful approach lets a deep neural network learn both feature extraction and the solution to the tackled problem jointly, by using raw unprocessed data as input. This type of approach will not be considered for this project, since we aim to leverage the mature conventional signal processing timbral feature extraction methods described in Section 3.1, which will help us better control the complexity of the system and gain insight into how the task is performed by the network by using handcrafted features.

# System Specification

In this section, we detail the specification of our timbral analysis system, narrowing the problem tackled to classification of piano sounds using spectrogram inputs, as well as crafting dataset used and the machine learning methods applied to construct the proposed system.

## Selection of input features: log-Mel spectrogram

Informed by the background research performed, the log-Mel spectrogram was selected at the start of project development as the signal processing timbral feature to be used as input feature for the timbral analysis system. This decision was made on the basis that the spectrogram gives a holistic view of temporal, spectral and harmonic characteristics in a single feature, making for a very compact representation of timbral information. Furthermore, the log-Mel spectrogram stands out from many of the other features studied in that it is perceptually motivated, as detailed in Section 3.1.7, and its characteristics are easily observed visually, making it more straightforward to interpret than other more numerical features. This choice was also supported by the precedent set by prior work performed on related tasks, namely [50] in which the log-Mel spectrogram is used as input to a CNN musical instrument classifier; this allows us to refer to this work as a point of inspiration and comparison as we develop our system.

However, we note that the numerous other features studied may provide better results when used in conjunction; but, within the constraints of this project, the spectrogram is expected to provide a detailed view of timbre without requiring development of multiple feature extraction systems. This allows us to more evenly divide project effort between development of feature extraction, pre-processing, and the development and analysis of a machine learning system.

## Problem specification: Binary classification of piano types using CNNs

In order to approach timbral analysis, we opted to construct a classification system as this is a common machine learning paradigm applied to frame perceptual tasks. For instance, a typical way of formulating a computer vision task is to perform image classification - machine learning methods are well-adapted to tackling such classification problems as detailed in Section 3.2.2. Furthermore, other formulations such as regression are less straightforward to apply to analysis of timbre, since timbre is a quality of sound as opposed to a measurable quantity. Formulating timbral analysis as a classification problem also allows us to apply a wide range of data to the project, since classification labels are readily available in a variety of musical instrument datasets, as reviewed in Appendix A – Review of isolated-note datasets. The simplest form of classification is binary classification, the two-class case, for which a network only needs to output a single, binary scalar value. We therefore select binary classification by timbre as the point of entry to development of our timbral analysis system.

Aiming to analyse more subtle timbral differences, to select the targeted labels to predict, we chose to look at classifying the type of instruments within the same family. Due to personal familiarity, as well as the wide availability of data relative to other instruments (see ), considering the timbral differences between different sorts of acoustic pianos was a natural choice. To target a specific label, we chose to specify the classification task to the discrimination between the two main types of acoustic piano: grand and upright. The difference in sonic character between these two classes of piano results from their differing sound production mechanisms, as well as the dimensions and shape of their bodies – grand pianos being larger and having strings laid out horizontally as opposed to upright pianos, whose strings are vertical in order to accommodate for their more compact size. Choosing to target these labels was strongly dependent on the availability of class-labelled data, our selection of targets being limited by which ground truth annotations are available in order to train and evaluate the classifier. Fortunately, these labels are commonly annotated since they are objective qualities of timbre (as opposed to subjective descriptors such as “bright” and “mellow” timbres) - in some data sources only the type or model of piano is specified, which is sufficient annotated information for our needs.

The final choice made in specifying the timbral classification system was selecting which type of machine learning method to apply to the specified problem. Given the choice of spectrograms as input feature to the system, we opted to apply convolutional neural networks, since these are naturally suited to learning local and “spatial” characteristics in such 2 dimensional maps. In particular, as explained in Section 3.2.2, CNNs are able to integrate the temporal as well as spectral aspects of a sound shown in a spectrogram, and their approximate translation invariance may allow for pitch-invariant inferences to be made. Additionally, deep CNNs are commonly applied to perceptual tasks (for instance in the field of computer vision), so they are an appropriate construct to apply in our attempt to emulate the human ability to perceive timbral differences between musical instruments. This is evidenced by this type of model’s prevalence in the literature applied to adjacent tasks such as MIR, especially when used in conjunction with spectrogram inputs, as cited in [24].

## Datasets assembled for the task

A key step in the specification of the proposed system was the selection and assembly of a dataset. In order to train, validate and test the piano classifier, we require a large dataset containing recordings of a range of acoustic pianos labelled by type (upright and grand), since these are our targets. We aim to collect a large amount and variety of training data for our classifier as this will improve its ability to generalise its capacity timbral inference to unseen examples. In particular, we highly value variety in terms of the recording and playing techniques, piano models, and subtypes represented in the dataset, so as to not overfit to any one of these variables which are likely to affect timbre but are independent of the class identification task at hand. In particular for pianos, these variables include the articulation (the playing technique used, e.g. use of the sustain pedal), the brand of instrument, body shapes and sizes, dynamics (the intensity with which the piano key is hit, e.g. pianissimo, forte, etc.).

### Assembly of the isolated-notes dataset using single-note piano samples

In our initial review of the data sources available (see Appendix A – Review of isolated-note datasets), we found multiple libraries of piano sounds recorded note-by-note. Many of these databases were assembled for use in music production as virtual instrument sample libraries (for more information on the format of virtual instrument databases, how they are created and applied, see [67]). These sample libraries capture a given instrument in high-quality note-by-note audio recordings, sometimes with several passes over the range of notes available on the instrument, at multiple dynamic levels (called velocity layers). While they were not created with research in mind, these detailed and carefully parametrised recordings present many desirable qualities for use in machine learning:

* Quantity: detailed libraries contain several samples for each of the 88-notes on the piano. Single, isolated note recordings ensure that the entire envelope of each note being played is captured in great detail in the dataset.
* Quality and variety: Recorded in high-quality in different environments, including close to anechoic “recording studio” conditions, or settings capturing the acoustics of a particular real-world setting (e.g. a concert hall). Each note of the instrument is played with different articulations (e.g. different velocities and sustain pedal usage). For music creation, these variations for each note are called sample layers, since they are meant to be assigned to the same key on a digital piano. For our purposes, we will treat each layer as a separate entry in a dataset.
* Labelling: the pitch, dynamics, articulation, type and sometimes model of piano recorded are labelled on a per-sample basis for accurate reproduction on digital pianos, making many forms of data analysis possible.
* Standardisation: virtual instruments are controlled using Musical Instrument Digital Interface (MIDI) technology, which means they adhere to common standards for pitch and velocity labelling.

We also found similar single-note sample databases intended for use in signal processing and music information research. Of these, we selected two which featured a large number and variety of piano instruments. The combination of multiple datasets provides a greater diversity of examples, types, recording conditions and sources of instruments, as well as a larger number of examples to support training and testing of a classifier.

It is important to highlight that restricting our system to considering only isolated-note data greatly restricts the considered timbral analysis task to a particular case, since the classifier will only ever be input monophonic data in which the instrument only plays one note at a time. Therefore, the system will not model more complex polyphonic effects which result from the interactions between multiple notes played simultaneously on the same instrument, for instance when a chord is played.

Given these considerations, we selected the 3 single-note upright and grand piano sample databases detailed in ***Table 4.3.1‑1***. These were openly available online in the case of *BiVib* and *MAPS*, and the *Nord Piano Library* was made available to us through access to a Nord Electro 6D 61 digital piano [68], graciously provided by a friend for our re-sampling experiments. These 3 data sources were favoured as they each contain examples of both classes of piano, are straightforward to adapt to our research purposes, and are comprehensive in terms of having multiple layers for all notes on the keyboard, yielding a large quantity and variety of data. Each data source containing examples of multiple instruments presents the additional advantage of grouping multiple pianos into the same format, reducing the effort required per-instrument to manually merge them into a master dataset and adapt them to our needs.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Dataset name** | **Author & Reference** | **Intended purpose** | **Timbral Annotations** | **No. & type of pianos** | **Layers sampled** | **Availability** | **Format** |
| ***BiVib*** | Papetti et al. [69] | Research | Dynamics  Type & Model  Recording conditions  Articulation | 1 Grand  1 Upright | - 10 velocities  - w/ & w/o sustain pedal  - 3 lid settings | Free, online | Fs = 96 kHz.  24 bits/sample.  Stereo.  wav file format. |
| ***MAPS*** | Telecom ParisTech [70] | Research | Dynamics  Type & Model  Recording conditions  Articulation | 5 Grand  2 Upright | - 3 velocities  - Recording proximity  - Sustain pedal randomly applied | Free, online | Fs = 44.1 kHz.  16 bits/sample.  Stereo.  wav file format. |
| ***Nord Piano Library*** | Clavia Nord AB [71] | Music Creation | Dynamics  Type & Model  Recording conditions  Articulation  Character | 12 Upright\*  12 Grand | - 3+ velocities  - w/ & w/o sustain pedal | Included with Nord digital pianos. | Sample rate & bit depth not specified\*\*.  Stereo. |

Table 4.3.1‑1: Data sources used to assemble the piano classification dataset.   
\*One of the 12 Nord upright pianos (“HonkyTonk Upright”) was excluded, as its sound was produced by replacing felt hammers with metal nails, therefore it is considered unrepresentative of piano timbre and left out of the dataset.  
\*\* Given that the samples in the library are pre-loaded to Nord digital pianos, their recording format is not specified, and therefore the sample rate and bit depth of these waveforms as used in our dataset is dictated by the re-sampling method implemented to capture the Nord samples, which is detailed in Section 7.2.

As detailed in ***Table 4.3.1‑1***, *BiVib* contains two sampled pianos, one grand and one upright, each with three lid positions (e.g. lid closed, lid open, lid removed for the grand piano). We considered each of these settings to be separate instruments for the purposes of our dataset, since these variations have a notable effect on the instrument timbre. Similarly, some of the pianos sampled in *MAPS* were recorded in two different settings: one close, in which microphones were placed in close proximity to the piano strings, and another ambient setting, for which the pianos were recorded from a greater distance so as to capture the sound perceived by a listener external to the player. Once again, both of these variations were kept as separate pianos in our dataset.

For the velocity layers, we used the full dynamic ranges provided (see the “Layers Sampled” column of ***Table 4.3.1‑1***) in all data sources except the *Nord Piano Library*, for which we re-sampled only a subset of the originally-sampled velocity layers, since the total potential number and ranges of the dynamic layers are not indicated by the manufacturer of this data source. Therefore, in order to capture several velocity layers most likely to come from distinct recordings for each note, we selected a sparse set of 3 evenly-spaced velocity layers, out of the possible 128 MIDI velocity levels the Nord Electro 6D digital piano accepts.

For the sustain pedal variable, which when depressed increases the resonance of the piano by releasing the dampers from all of its strings, we omit the use of the sustain pedal in the case of the *Nord* samples in order to simplify the re-sampling process, and similarly discard the sustained samples provided in the *BiVib* dataset to reduce usage of limited storage resources for the combined dataset. We note that both sustain pedal “on” and “off” layers are only fully sampled in the *BiVib* dataset and not in the other two sources, so retaining both of these layers would serve only to increase the quantity, generality and diversity of the dataset, as we do not aim to explicitly model the effects of the sustain pedal on piano timbre.

The size of the dataset assembled was limited by the available storage and memory, which was a determining factor in selecting only three databases to form our combined dataset. We further constrained the dataset by restricting the note range to the two central octaves of the piano, from one octave below middle-C to one octave above. This reduced the number of notes featured in the dataset to 25 notes per velocity layer per instrument, but hopefully allows the timbral classifier trained on this smaller note range to perform well despite the discarding of the bass and treble extremes of the piano keyboard. Note that we would expect a classifier making use of the full piano note range to perform better at the task of discriminating between upright and grand pianos, since it may build a more complete image of the differences between these piano types across a wider range of pitches.

Given these restrictions, the isolated-notes dataset assembled for our piano classifier contains 37 instruments in total (of which 32 are unique pianos), giving a sum of 3825 isolated-notes dataset entries overall.

### Generation of the melody dataset: alternative dataset derived from single-note data

Use of these single-note databases gives us a degree of flexibility beyond analysis of isolated, single note signals, since the data can be repurposed to simulate monophonic or even polyphonic pieces being played on the sampled pianos. This motivated the creation of an alternative dataset, derived from the single-note data, made up of segments drawn from melodies generated using the sampled piano notes. Using such a dataset, the classifier is input on a portion of a melody "played" on one of the virtual instruments, instead of making a decision based on a single isolated note as is the case when the single-note data is used as input.

Assembly of this dataset required sourcing a data source containing a large variety of monophonic melodies that we can apply to our sampled instruments using a sequencer. For this purpose, we found the *mono-midi-transposition-dataset* [72], an extensive library of almost 16,000 monophonic melodies encoded in the MIDI file format used to represent musical pieces in a digital form. Using the note-sequence information in these MIDI files, we sequence the piano samples corresponding to a given instrument (drawn from the isolated-notes dataset) with a fixed velocity to emulate the melodies being played on the sampled instrument. Given the restriction of the isolated-notes dataset to the two central octaves of the piano keyboard, we also limit the sequenced melodies to this note range.

Using this method, we assemble the melody dataset using 20 unique melodies per sampled velocity layer available for each instrument. For simplicity, melody sequencing is applied using a fixed velocity within each melody, so that all notes are played at the same intensity given by the considered velocity layer. For *BiVib*, we group the 10 sampled velocities into 3 different levels roughly corresponding to *piano, mezzo,* and *forte*, and for the other two libraries we use the 3 available layers, such that an equal number of 3\*20 melodies is generated for each instrument in the dataset. This results in 60 unique melodies being applied to each instrument from the isolated-notes dataset. This yields 2220 melodies in total across all of the instruments. In this process, we ensure that unique melodies are applied to each velocity layer and each instrument to prevent the classifier from learning melodic characteristics instead of timbral aspects for instrument prediction using this dataset.

The number of 20 melodies per sampled layer was selected so as to provide a substantially larger dataset than the original in terms of the raw number of data samples, while further increasing the number of melodies would saturate the available storage and memory budget while likely providing diminishing returns. This can be seen as a form of data augmentation, as it greatly increases the number of entries in the dataset. However, it does not increase the variety or quantity of unique data points in the dataset in terms of the number of actual instruments are featured, since we are applying multiple melodies to each piano.

The principal objective for the use of this alternative dataset, as opposed to direct application of the original isolated-notes dataset, is the fact that this frames the classification as a more realistic task given since it emulates the real-world scenario of a human playing a melody on the piano. This approach also provides more musical context to learn from since multiple notes are contained in each dataset entry, and allows for a decision to be made on the basis of more than one note at a time. However, a possible caveat of using this type of data is that the simulated playing sounds somewhat unnatural. This is likely due to the fact that the sounds of the piano keys being released after a note is played are not taken into account by our melody sequencer, as well as the fact that we use a single uniform velocity to play each melody, whereas dynamic variation is an important sonic feature of real-world piano playing.

From this point, we will thus consider two related but separate piano classification tasks: the isolated-notes timbre classifier, which is trained and makes decisions on a per-note sample basis using the original isolated-notes dataset; and the melody timbre classifier, which is instead input elements drawn from the derived melody dataset. We will tailor subsequent pre-processing steps and approach to CNN design to each of these tasks separately as different problems, before comparing the two approaches to piano timbre classification.

# Design and analysis

In this section, we describe the design considerations we applied to the system, detailing the critical choices made at each step for the selection of datasets, pre-processing pipeline, architecture, and training of our timbral classifier.

## Pre-processing Methods

### Processing of raw single-note waveforms

Before further processing and feature extraction, the single-note audio waveforms are processed in order to homogenise their non-timbral aspects, such as format, volume and duration.

Each of the single-note recordings in the used data sources were captured by placing 2 microphones, generally either side of the centre of the piano aiming towards the soundboard, so as to replicate the pianist’s listening position. While these stereo signals provide realistic spatial information and capture the pianos with a high-fidelity to how they sound to a human player in a physical space, our processing pipeline was designed with monophonic signals in mind, since spatial information is not directly relevant to analysing the timbre of a particular instrument. This restriction reduces the computational complexity, memory and storage requirements of subsequent processing by a factor of 2 while retaining the recorded timbre, albeit at a lesser level of detail. Therefore we opted to discard spatial audio information by casting the stereophonic recordings to 1-dimensional mono waveforms.This was initially achieved by summing the left and right channels. However, we found upon listening to the summed waveforms that this gave poor qualitative results, due to the partial phase cancellation effects on the signal’s spectrum that result from combining recordings of the same source made from different positions [73]. As the used data sources did not detail the specifics of their microphone placements, we were not able to systematically compensate for this interference in order to preserve the spectral information of the recordings. Therefore we opted instead to separate each of the recorded channels, retaining only the left channel in order to limit the dataset’s storage and memory requirements, while preserving data variety in terms of the number of notes, layers, recording environments, and instruments featured in the dataset.

The waveforms are then resampled, if necessary, to the common 44.1 kHz sample rate selected for the combined dataset. This was selected as it is a common standard for high-quality digital music formats such as Compact Disc (CD) Digital Audio [74], ensuring that the Nyquist frequency of 22.05 kHz is beyond the limit of human hearing to prevent audible quantization artifacts. This was also a beneficial choice since 44.1 kHz was the lowest sample rate used in the original data sources used (see the “format” column in ***Table 4.3.1‑1***), meaning up-sampling is not required to achieve this. Finally, each waveform is amplitude-normalised in order to equalise each note’s volume, after having removed any DC offset present in the monophonic signals. The normalisation is performed so as to scale the waveform peak to the maximum value allowed by its bit-depth, i.e. 16-bit integer for MAPS and 32-bit integer for BiVib and Nord, in order to preserve detail and utilise the full dynamic range of these datatypes. This allows the processed waveforms to retain a CD-equivalent minimum quality standard (16-bit precision [74]), while being stored in reasonably compact datatypes. Each waveform was cropped to a length of approximately 2.21 seconds so that each sample in the dataset is an array of the same dimensions, selected to match the minimum-duration isolated-note sample seen in the dataset so that no trailing silences are kept. We also ensure, using the available annotations, that no leading silences are kept in the processed single-note waveforms, loading only the portions of the recordings contained between the onset and release of the note, as indicated in the accompanying annotations supplied with each data source.

The labelled class annotations “upright” and “grand” are converted to binary values 0 and 1 respectively, so that the label error between predictions and ground truth can be measured numerically for training and evaluation of the classifier. Therefore “positive” binary annotations refer to grand pianos and “negatives” to upright pianos, which is an arbitrary labelling choice which should not influence how the binary classifier considers each class.

### Feature extraction: generation of Mel spectrograms

In order to generate Mel spectrograms from the raw audio waveforms, several parameters must be selected as described in sections 3.1.4 as pertains to the spectrogram, and 3.1.7 concerning conversion to the Mel scale. In the design of our pre-processing pipeline, we initially selected values for these variables based on those typically applied for speech processing/MIR in the literature, notably taking inspiration from closely related works such as [22].

The frame length selected was 25 ms, which is slightly longer than the window length of 10 to 20 ms typically used in speech processing [35], as we expect the timbre of a piano to evolve slower than a typical speech signal, at least during the steady state portion of a single-note sound. However, this longer analysis frame allows for a higher frequency-resolution spectrogram, as dictated by the time-frequency resolution trade-off discussed in Section 3.1.4.

A hop size of 10 ms between frames was chosen, so that roughly ½ of each frame overlaps with the next. This allows for a high effective time resolution on the x-axis of the spectrogram.

Each frame was zero-padded in order to achieve frequency interpolation as detailed in Section 3.1.4, resulting in a 2048-sample long STFT window (a duration of roughly 46 ms at the 44.1 kHz data sample rate). Making the window length a power of 2 allows divide-and-conquer based STFT algorithms such as the Fast Fourier Transform to perform optimally. The window function used in the application of the STFT is the Hamming window , selected as such to minimise spectral leakage effects.

The considered frequency range for the y-axis of the spectrogram is selected as 20 to 20,000 Hz, corresponding to the human hearing range, so as to retain all perceivable information contained in the audio signals.

The number of Mel filters in the bank used to apply the Mel scale was selected as 300, as this provided favourable results heuristically by maximising the detail in the y-axis of the spectrogram, while preventing the creation of empty frequency bins.   
*TODO: Add comparison of aspect of spectrograms using different values of n\_mels for single-note waveforms, justifying that n\_mels=300 provided the highest-detail results.*

The magnitudes in the Mel-spectrograms are then re-scaled to the decibel scale, in order to reflect human perception of volume as motivated in Section 3.1.7.

This results in 300x221 maps.

*TODO: Add example spectrogram plots generated from the single-note samples. Show different notes on uprights and grands, showcasing the differences between the two instruments’ characteristic Mel-spectrograms and the variations from one pitch to another. Perhaps for each piano, show a 3x2 set of plots with 2 different notes and 3 different velocity layers for each note.*

### Isolated-notes-specific spectrogram processing

Once spectrograms of the single-note piano sounds are generated, we apply additional processing before input to the CNN classifier. Namely, we normalise the input feature space, which is a common strategy to increase training performance as mentioned in Section 3.2.2. This aims to make each element of the input maps (spectrograms) occupy a similar distribution of values so as to be considered with equal importance by the CNN’s input layers at the start of the training process. In order to apply this to the magnitudes in each time-frequency entry in a given single-note spectrogram, we initially trialled two forms of spectrogram normalisation: statistical normalisation, and normalisation using the estimated energy of the fundamental frequency of the note pitch.

The statistical approach standardises (also known as computing z-scores) the distribution of magnitudes in a given spectrogram’s entries in order to achieve a zero-mean unit variance distribution. This is applied by subtracting the mean from each entry, and dividing by the standard deviation of the values contained in the spectrogram as detailed in

.

where and are the magnitude at spectrogram entry and the corresponding standardised score respectively, and and are the mean and standard deviation computed over all frequencies and time frames in spectrogram.

The other approach to spectrogram normalisation is using the magnitude of the fundamental, which is achieved by dividing by the estimated energy contained in the spectrogram bin closest to the fundamental frequency. This quantity is estimated by computing the peak magnitude (over the duration of the spectrogram’s time-axis) of the Mel frequency bin whose centre frequency is closest to the known fundamental frequency of the note. The fundamental pitch is given by the pitch annotation supplied for each note in the dataset. The motivation for this approach is to render the spectrogram’s magnitude distribution less dependent on the intensity with which the note is played, since this is likely to dictate the energy of the fundamental pitch.

Comparing these two spectrogram normalisation modes, we decided heuristically to opt for standardisation, as this takes into account the statistics of a given spectrogram, as opposed to simply dividing by a scalar in the case of normalisation by the fundamental magnitude. The latter being a simpler operation on the spectrogram’s magnitudes, it is more likely to be easily learnt by the early layers of the CNN than statistics-based normalisation. Furthermore, the latter approach is subject to imprecision due to the width and location of frequency bins likely not being aligned with note fundamental frequencies. Thus, the peak magnitude in the considered frequency bin is likely to provide an erroneous estimate of the note’s intensity.

*TODO: Give the number of samples in the dataset, specifying what constitutes one sample.*

### Melody-specific processing

For the second dataset made up of monophonic melodies generated using sequencing, additional considerations apply to the pre-processing of waveforms and spectrograms. We first must take steps to ensure that each note in a sequence sounds even, and that the generated melodies sound as natural as possible. For this purpose, we again require the single-note waveforms to each be amplitude-normalised as detailed in Section 5.1.1, since once sequenced in a melody, the discrepancy in volume between consecutive notes makes for unnatural-sounding melodies, with an uneven emphasis on each note.

Before inserting a single-note sample to a melody, a slight envelope is also applied to the note’s waveform to prevent clicks resulting from sudden changes in amplitude between notes. We apply short concave fade in and out functions to the start and end of the sample’s waveform respectively, in order to shape its attack and release. This is achieved by multiplying the first 5 ms of the single-note waveform with a square-root function whose gain runs from 0 to 1 over this duration, and conversely multiplying the last 5 ms of the waveform with a reversed square-root function going from gain 1 to 0. These functions and their effect on the single-note waveforms are visualised in ***Figure 5.1‑1*** and ***Figure 5.1‑2*** respectively.

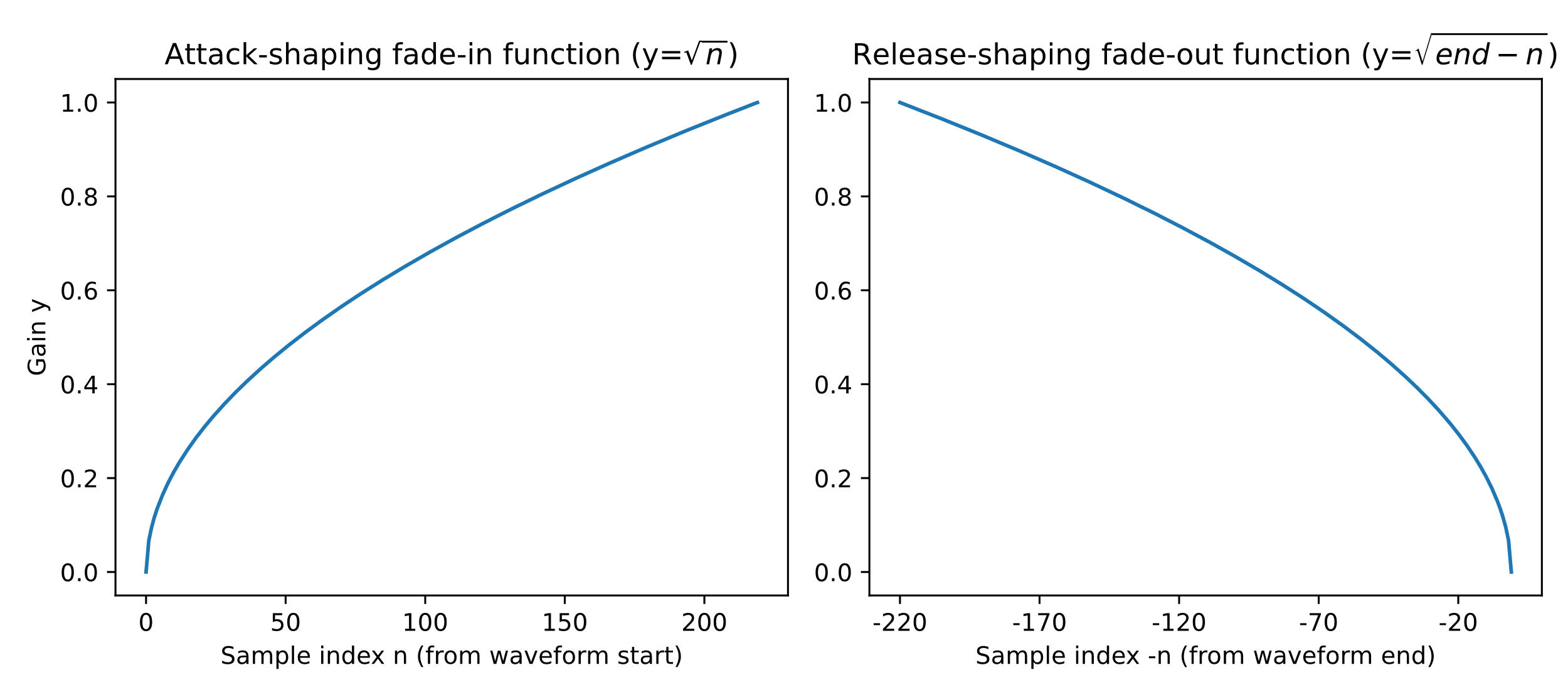


Figure 5.1‑1: Envelope-shaping functions applied to the start and end of each single note waveform before use in a sequenced melody.

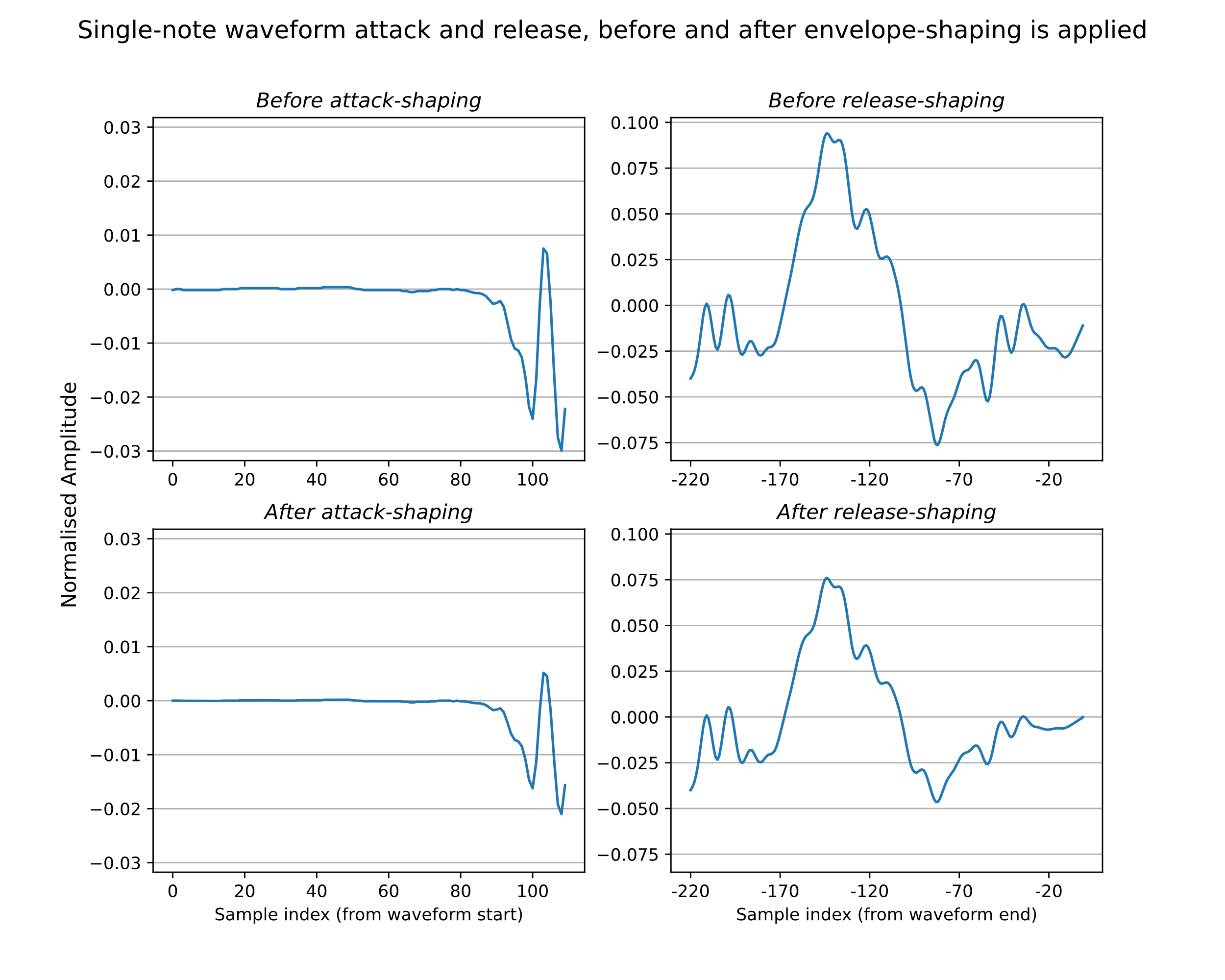


Figure 5.1‑2: MAPS single-note sample MAPS\_ISOL\_NO\_F\_S1\_M60\_AkPnBcht (instrument AkPnBcht, MIDI pitch 60, forte velocity). The horizontal gridlines show the amplitude of the “after” waveforms increasing or decreasing progressively relative to the “before” waveforms, in accordance with attack or release envelope-shaping functions applied.

Attack and release durations of 5 ms, as well as the concave envelope shapes, were selected heuristically for this purpose, as these are typically used by sample instrument creators to achieve realistic-sounding results for sequenced piano sounds, as detailed in [75]. These fades are kept as short as possible in order to retain the characteristics of the attack and steady-state timbre of the instrument itself.

Once a melody is sequenced with a particular sample instrument and its spectrogram is generated as detailed in Section 5.1.2, we draw non-overlapping segments from the melody, each of which constitutes one dataset sample. For this we must select the duration of one of these portions, i.e. the time-length of each spectrogram in the melody-based dataset. If the spectrogram is too long, an overly complex CNN would be required to capture the finer-timescale aspects of the instrument’s timbre, while if the spectrogram is shorter than the length of a single note in the melody, the task reduces to the same problem as identifying the instrument from the isolated-notes dataset. Informed by these heuristic considerations, we select the length of these segments using the average number of notes per unit of time in the considered melodies. Across the MIDI melody dataset, the average length of a note is approximately 0.3 seconds.Therefore, selecting a melody segment length of 2.21 s yields an average of about 7 notes per spectrogram, which should allow for a balance between time-axis detail and providing multiple-note context. Furthermore, this value matches the dimensions of the spectrograms computed for the isolated-notes dataset, which allows us to apply identical CNN structures to either dataset, enabling easy comparison and adaptation between the two considered tasks during CNN development.

*TODO: Add example spectrogram plots for the generated melodies. Show one plot with a full melody, and one with a single segment drawn from the melody to show what 1 sample of the dataset looks like.*

As is the case for the isolated-notes data, we apply standardisation to the magnitudes in each spectrogram as a feature normalisation step. Direct application of the alternative fundamental-magnitude-based normalisation is not possible in the case of the melody segments, since these contain multiple pitches and therefore multiple fundamentals.

*TODO: Give the number of samples in the dataset, specifying what one sample constitutes.*

## Dataset partitioning for classifier training and evaluation

From this point, we refer to a dataset “sample” as a single entry to the considered dataset, containing a pre-processed Mel-spectrogram as the network input feature and the associated ground-truth annotations, as well as metadata such as the note velocity, the name of the instrument used to generate the sound, and the name of the source database from which it was drawn. This terminology is not to be confused with musical “samples” in the context sample libraries/instruments, which refer recordings of an isolated note as was employed in Section 4.3. Nevertheless, in the case of the isolated-notes dataset, each entry indeed corresponds to a sampled isolated piano note drawn from one of the data sources; meanwhile, each entry in the melody dataset has a spectrogram that is likely to contain several single-note samples.

In order to train, validate, and test classifiers to discriminate between the two targeted classes of piano sounds, we need to establish independent training, validation and testing partitions (see Section 3.2.2 - *Common training and data considerations for neural networks*) of the isolated-notes and melody datasets described in Section 4.3. In this sub-section, we describe the approaches applied to partitioning the isolated-notes and melody datasets for model training and performance evaluation, by sorting the samples into training, validation and testing subsets using various schemes. We then detail the final methodology used for splitting the dataset into these subsets for our system, explaining the reasoning behind the approach.

The first important consideration for this process is preventing the introduction artificial bias during model training and selection. This involves ensuring that the development subsets (i.e. the training and validation sets) are as class-balanced as possible, each of the two targeted piano classes being sufficiently represented in the training set to train an unbiased classifier. This is particularly difficult to achieve since the master dataset itself unbalanced, containing 20 “grand” instruments and 17 “upright” instruments. Meanwhile, if either the validation or test sets are unbalanced, special care must be taken to score model predictions so as to give a balanced view of the binary classifier to discriminate between the two classes, as will be detailed in Section 6.1.1. Minimising development set bias also involves ensuring models under development are exposed, via the makeup of the training and validation subsets, to a variety of note pitches or melodies, of articulations (e.g. velocities), of data sources, and of recording conditions, which should help us to create and select models which generalise well to diverse data.

The second consideration applied to dataset partitioning is the segregation of the three subsets. If samples drawn from the same instrument and data source appear across several of the subsets, then these subsets cannot be considered as entirely independent. Namely, correlations extraneous to the targeted timbral information may exist between such samples resulting from their shared origin, which models could take advantage of in order to “cheat” when making predictions on the validation and test sets. Segregation is particularly crucial between the development and test partitions, as we want the test data to be a truly unseen set so as to gain an objective understanding of an evaluated model’s performance and ability to generalise.

Given these balancing, variety and segregation criteria, we developed and compared several dataset-splitting schemes:

* *Mixed partitioning* (randomised), which randomly draws samples from the full dataset, without replacement, to build each data subset. This approach maximises the potential variety and average class-balance of each subset, but does not achieve any segregation of data sources and instruments between the subsets.
* *Partitioning by velocity layer* (deterministic), which splits the master dataset into three subsets of equal size using the three velocity groups featured in the dataset (*piano*, *mezzo* and *forte*). Since all three velocity layers exist for each note of each instrument, this achieves a fair degree of variety, and a class-balance equivalent to that of the master dataset in each of the subsets, but also means that each note of each instrument is present in all subsets. Therefore, this method does not produce segregated subsets beyond the separation of velocity layers, which may allow for a classifier to “cheat” during evaluation.
* *Partitioning by instrument* (randomised), which builds subsets by randomly drawing all of the samples belonging to an instrument at once. This can be optimised for balance by alternately selecting pianos from each class when drawing a new instrument to add to each subset. The subsets generated using this method are therefore partially segregated and mostly balanced, and also benefit from a high degree of variety.
* *Partitioning by data source* (deterministic), which splits the dataset’s samples according to their original database (i.e. *MAPS*, *BiVib* or *Nord Piano Library*). This maximises the degree of segregation achieved between subsets, but results in a trade-off in the form of decreased variety of the subsets. Furthermore, only those subset(s) made up of *BiVib* and/or the *Nord Piano Library* are close to class-balanced.

Given these considerations, we settled on a hybrid approach combining dataset partition by instrument and partition by data source. Within the development set, the splitting between training and validation subsets is performed using balanced partitioning by instrument, and the testing/development split is partitioned by data source. The *Nord Piano Library* is used for the development set, while *BiVib* and *MAPS* jointly form the test set. This partitioning scheme allows us to develop our models using fairly heterogeneous, balanced and segregated training and validation sets, and evaluate them using a genuinely unseen held-out test set by using samples drawn from a separate data source. Also, using this scheme, the partitioning by instrument of the development set can be randomised to prevent repeated creation of the same split for every training/validation run, so as to mitigate the risk of overfitting to one particular choice of subsets during development. Furthermore, the training subset is shuffled every time training is performed, to prevent each model from being exposed to the same sequence of inputs which provides a different training signal to each trained model.

*TODO: Add tables listing each instrument in each of the development and test partitions, the number of samples for each instrument and in total for each partition (for both the isolated-notes and melody datasets), and the instrument classes.*

## CNN Architecture and Training

### CNN architecture design

#### Initial architecture: TimbreCNN

Our initial CNN design was a simple fully-convolutional 5-layer architecture based on those previously implemented for computer vision tasks in lecture courses studied, namely the fully convolutional variant of LeNet introduced in [76]. Following this simple guide architecture, we designed a 5-convolutional-layer network, with increasing numbers of filters per convolutional layer as the inputs reach deeper into the network, then reducing towards the output. This results in a roughly symmetrical architecture, with the majority of the trainable parameters in the middle hidden layers of the network. This makes for a much deeper network than LeNet, a decision made to reflect the more complex task tackled by our CNN as compared to LeNet’s character recognition using low-resolution input images. Indeed, a more expressive model is likely to be needed to infer the more intricate patterns in the spectrograms used in our application. The proposed architecture has 46,328 trainable parameters, as detailed in ***Table 5.3.1‑1***.

In the proposed architecture, we iterated upon the FCNN variant of LeNet by adding Batch Normalisation layers. This is a common addition in modern deep CNNs, which we have found to provide notable training speed improvements in prior experiments on computer vision, as motivated in Section 3.2.2 under *Common training and data considerations for neural networks*. This network architecture, defined in ***Table 5.3.1‑1***, is made up of 5 blocks followed by an output layer. Each building block has a similar structure, containing a 5x5 convolution, followed by a batch normalisation, followed by a max-pooling layer and finally a ReLU activation function.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Block** | **Name** | **Type** | **Operation Dimensions** | **Output map dimensions** | **Num. Learnable Parameters** |
| *In* | *Input data (spectrogram)* | Network input | - | 1x300x222 | - |
| *1* | *Conv2d-1* | Convolutional | 6x5x5 | 6x296x218 | 150 |
| *BatchNorm2d-1* | Batch Normalisation | 6 x all | *same* | 12 |
| *MaxPool2d-1* | Maximum-pooling | 1x2x2 | 6x148x109 | - |
| *ReLU-1* | ReLU Activation | all | *same* | - |
| *2* | *Conv2d-2* | Convolutional | 18x5x5 | 18x144x105 | 2700 |
| *BatchNorm2d-2* | Batch Normalisation | 18 x all | *same* | 36 |
| *MaxPool2d-2* | Maximum-pooling | 1x2x2 | 18x72x52 | - |
| *ReLU-2* | ReLU Activation | all | *same* | - |
| *3* | *Conv2d-3* | Convolutional | 45x5x5 | 45x68x48 | 20250 |
| *BatchNorm2d-3* | Batch Normalisation | 45 x all | *same* | 90 |
| *MaxPool2d-3* | Maximum-pooling | 1x2x2 | 45x34x24 | - |
| *ReLU-3* | ReLU Activation | all | *same* | - |
| *4* | *Conv2d-4* | Convolutional | 18x5x5 | 18x30x20 | 20250 |
| *BatchNorm2d-4* | Batch Normalisation | 18 x all | *same* | 36 |
| *MaxPool2d-4* | Maximum-pooling | 1x2x2 | 18x15x10 | - |
| *ReLU-4* | ReLU Activation | all | *same* | - |
| *5* | *Conv2d-5* | Convolutional | 6x5x5 | 6x11x6 | 2700 |
| *BatchNorm2d-5* | Batch Normalisation | 6 x all | *same* | 12 |
| *MaxPool2d-5* | Maximum-pooling | 1x2x2 | 6x5x3 | - |
| *ReLU-5* | ReLU Activation | all | *same* | - |
| *Out* | *Conv2d-Out* | Convolutional | 1x5x3 | 1x1x1 (scalar) | 90 |
| *BatchNorm2d-Out* | Batch Normalisation | 1x1x1 (scalar) | 1x1x1 (scalar) | 2 |
| *Sigmoid-Out* | Sigmoid Activation | 1x1x1 (scalar) | 1x1x1 (scalar) | - |

Table 5.3.1‑1: Details of each layer in the initial CNN architecture applied.   
The architecture has 46,328 total trainable parameters.  
Map and filter dimensions are denoted using the standard convention of channels x height x width. The number of channels is the depth of the layer/filter, height is the span over the frequency-axis and width concerns the time axis. The input map for each element in the table rows is the previous row’s output.

The dimensions of the convolutional layers in each block are selected to achieve the required downscaling of feature maps from the input size to the scalar output in six convolutions (paired with down-sampling by max-pooling). The series of 5x5 convolutions allows the network to learn patterns spanning increasingly wide portions of the time-frequency map as we go deeper (from input to output) into the network. This allows the architecture to make inferences at a variety of scales and levels of abstraction from the input spectrograms. In particular, the early layers can capture smaller time and frequency-scale details in the spectrograms. For instance, the first layer (Conv2d-1) has a receptive field of 5x5 in the input spectrograms, which corresponds to 5 out of the 300 Mel frequency bins (i.e. a narrow portion of the frequency range at low-frequencies, and larger regions of the spectrum at high frequencies) and a 50 ms span on the time axis. Layers further from the input, such as Conv2d-3, will therefore operate over larger receptive fields drawn from the input spectrogram. In general, the number of learnable parameters of a convolutional layer is given by its filter operating dimensions multiplied by the number of channels in the layer’s input map – these values are computed in the rightmost row of ***Table 5.3.1‑1***. Added biases are not included in the learnable parameters for the convolutional layers in this architecture, since these are made redundant by the added bias of the batch normalisation layers that follow. Instead, in convolutional layers, only the kernel weights of each convolution filter are optimised.

Batch normalisation is applied over all of the pixels in a BatchNorm layer’s input map on a per-channel basis, hence these layers produce output maps of the same dimensions as their inputs. A batch normalisation layer has two learnable parameters for every channel in the layer’s input map – one weight and one bias. The use of this layer in each block injects a degree of regularisation into the training process in each stage of the network, which is thought to help prevent overfitting in such deep CNNs.

The maximum pooling layers are the key to the network’s dimensionality reduction, as they allow feature maps to be progressively reduced from the high dimensional input space to the scalar prediction value. At the end of each block, the each dimension in the feature maps is reduced by a factor of 2 by retaining only the maximum values in each non-overlapping 2x2 image patch. Max-pooling layers also enable the CNN to operate with local translation invariance, meaning patterns can be recognised even if their locations in the time-frequency map vary from one sample to the next. This should help the network to learn a degree of pitch and onset-timing invariance for the timbral patterns it analyses.

The activation functions introduce non-linearities in each block so that more complex transformations can be learnt. We use ReLU to prevent the issue of vanishing gradients, since this is a deep architecture which may stall if gradient-vanishing-prone activations were used, as motivated in Section 3.2.2. Activation functions are computed entry-wise on their input feature maps, hence the feature map dimensions input to the activations are preserved in their output.

The final layer placed after the 5 main network blocks, which reduces the feature map size to a scalar, uses an appropriately-sized single-channel convolution so as to form a 1x1x1 output map. This is followed by batch normalisation, and a Sigmoid activation function. This activation was selected as it is a standard choice for the output stage of binary classification, thanks to its favourable properties for this type of task. Namely, the Sigmoid activation transforms its inputs to values in the range [0, 1], which can be interpreted as the network’s predicted probability of a sample’s belonging to class 1 (the “upright” label), and allows for a threshold of 0.5 to be used to convert network output to a binary prediction.

*TODO: Describe how this architecture is used for CNNs used to tackle both tasks: SingleNoteTimbreCNN and MelodyTimbreCNN. They differ in their training hyperparameters.*

#### Smaller and narrower architecture: TimbreCNNSmall

*TODO: Describe the second iteration, deeper and narrower architecture with fewer parameters (10906)*

To iterate upon this initial architecture, we studied several more specific architectures tailored to timbral or musical analysis of spectrograms. Namely, having based our approach to spectrogram processing on that presented in [22], we reviewed the proposed CNN architecture for timbral instrument classification in the accompanying source code published by the author [77] (attempts to reproduce the referenced work’s results using the models and data supplied were unfortunately unsuccessful). We also considered the more general guidelines cited in [24], which presents common successful paradigms for the design of CNNs tackling time-varying musical-domain problems with spectrogram inputs. One such strategy adopted in [78] is to explicitly render the CNN pitch-invariant by using larger max-pooling and convolutional kernels over the frequency axis than the time axis (using tall rectangular kernel shapes instead of the typical square kernels).

In our second architecture design, we aimed to emulate this musically-motivated approach by using taller maximum pooling kernels, while reducing the number of parameters with a narrower architecture, using a series of smaller 3x3 convolutions. As is detailed in ***Table 5.3.1‑2***, this smaller architecture features four hidden convolutional blocks as opposed to the initial architecture’s five, so it is a more compact architecture with fewer overall convolutional layers. We also add a max pooling layer to the output block which aims to combine temporal information over the layer input’s 4 entries along the time-axis, so that only the most important condensed temporal information for characterising the piano timbre is retained for the output.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Block** | **Name** | **Type** | **Operation Dimensions** | **Output map dimensions** | **Num. Learnable Parameters** |
| *In* | *Input data (spectrogram)* | Network input | - | 1x300x222 | - |
| *1* | *Conv2d-1* | Convolutional | 8x3x3 | 8x298x220 | 72 |
| *BatchNorm2d-1* | Batch Normalisation | 8 x all | *same* | 16 |
| *MaxPool2d-1* | Maximum-pooling | 1x2x2 | 8x149x110 | - |
| *ReLU-1* | ReLU Activation | all | *same* | - |
| *2* | *Conv2d-2* | Convolutional | 16x3x3 | 16x147x108 | 1152 |
| *BatchNorm2d-2* | Batch Normalisation | 16 x all | *same* | 32 |
| *MaxPool2d-2* | Maximum-pooling | 1x3x2 | 16x49x54 | - |
| *ReLU-2* | ReLU Activation | all | *same* | - |
| *3* | *Conv2d-3* | Convolutional | 32x3x3 | 32x47x52 | 4608 |
| *BatchNorm2d-3* | Batch Normalisation | 32 x all | *same* | 64 |
| *MaxPool2d-3* | Maximum-pooling | 1x3x2 | 32x15x26 | - |
| *ReLU-3* | ReLU Activation | all | *same* | - |
| *4* | *Conv2d-4* | Convolutional | 16x3x3 | 16x13x24 | 4608 |
| *BatchNorm2d-4* | Batch Normalisation | 16 x all | *same* | 32 |
| *MaxPool2d-4* | Maximum-pooling | 1x3x3 | 16x4x8 | - |
| *ReLU-4* | ReLU Activation | all | *same* | - |
| *Out* | *Conv2d-Out* | Convolutional | 1x4x5 | 1x1x4 | 320 |
| *BatchNorm2d-Out* | Batch Normalisation | 1 x all | *same* | 2 |
| *MaxPool2d-Out* | Maximum-pooling | 1x1x4 | 1x1x1 (scalar) | - |
| *Sigmoid-Out* | Sigmoid Activation | 1x1x1 (scalar) | 1x1x1 (scalar) | - |

Table 5.3.1‑2: Details of each layer in the smaller and narrower second CNN architecture.   
The architecture has 10,906 total trainable parameters.

The general motivation for this architecture is to improve upon the initial one by using musical-domain intuition so as to tailor the network’s structure for timbral analysis from the Mel-spectrogram. A less complex architecture can therefore be used, since more informed approach to CNN design should prevent the network from being over-specified.

### The training process and hyperparameters

In our system design, the proposed CNNs were trained using mini-batch gradient optimisation with the Adam optimiser algorithm, which is a prevalent scheme for training deep CNNs [79] and therefore was a standard choice for the purposes of our experiments. Similarly, the loss function used to measure the network’s prediction error in training was fixed as Binary Cross Entropy, the formula for which is shown in . This is a particular case of Cross Entropy loss, which was introduced in , applied to a 2-class problem – thus, it is a generally advised choice of loss function for binary classification tasks.

In the expression given for Binary Cross Entropy loss in , is the network’s prediction output, is the ground truth reference label, and , are the class labels considered. In our system, these two classes are “grand” and “upright” with binary labels and respectively.

The main hyperparameters which were considered as variables in our experiments to regulate the training process are defined in Section 3.2.2 - *Common training and data considerations for neural networks*. The motivations and considerations we applied to tuning each of them are as follow:

* The number of *epochs* is the number of optimisation passes performed over the training set. During each epoch, the model is exposed to the entire training set in batches. Increasing the *epoch* number should increase how closely the training set is fit by a model, since it controls how long the model trains for.
* The *Batch size* sets the number of samples per batch of data. Each batch is a subset of the training set drawn at random, over which the optimiser performs one optimisation step (error calculation, backpropagation, and updating of all network parameters). The smallest possible batch size is 1, which equates to updating parameters using a single dataset sample at a time. This makes the update steps noisy, since they are susceptible to a single data point’s fluctuations. Therefore, increasing the batch size means each optimisation step is made on the basis of a more significant subset of the training data – thus, intuitively, larger batch sizes stabilise the learning process yielding smoother convergence. However, the full training set cannot be used for each batch, as this is computationally inefficient as detailed in [61]. Therefore, the selection of a batch size is constrained by the computational resources available, and we will not be able to explore the use of batch sizes larger than 512 samples our experiments.
* The optimiser’s *Learning rate* controls the size of each optimisation step taken. In general for optimisation algorithms, a large step size can increase the speed of convergence towards a local minimum by making the training process more aggressive, but can also cause the training process to over-shoot optima without having a fine enough step to converge to satisfactory model parameters. Therefore we may increase the learning rate to achieve a faster converging model, but be careful to prevent the training process from stalling if it is too large. Furthermore, each optimisation step is informed only by the considered data batch for that step – as a result, increasing the learning rate increases the influence of each batch of data on the parameters. To take this fact into account in our mini-batch optimisation scheme, a strategy we applied when tuning these hyperparameters is to increase the batch size in conjunction with the learning rate, such that larger optimisation steps can be informed by a more statistically representative sample of the dataset.

These considerations, which concern tuning the training process, have a large influence on the models’ training behaviour. Therefore, the set of hyperparameter values used is a crucial aspect of model selection beyond selecting an architecture. The methodology that we designed to search for an optimal set of these hyperparameters for each of the considered classification systems is outlined in Section 6.1.2, in our discussion of the model selection process.

# Model selection and evaluation methodology

### Scoring the classifier

In order to evaluate a given trained model’s performance on the classification task, using a validation or test set, we need to select appropriate scoring metrics to quantify its performance and compare it against other models. We carefully selected metrics adapted to the tackled problem, aiming to give a comprehensive and unbiased measure of a model’s piano classification performance, taking into account the distributions of our validation and testing datasets. In this sub-section, we describe the choices made for our evaluation of a piano classifier’s performance, comparing several metrics before selecting those most insightful for the considered problem.

At test or validation time (when the network makes predictions without optimising), evaluation is performed on a sample-by-sample basis, where one sample contains a single input spectrogram. For instance, for the isolated-notes dataset and classifier, this means making a prediction based on the spectrogram of a single note drawn from a particular instrument’s samples. Similarly, for the melody-based classifier, the network makes a prediction on the spectrogram of a single segment drawn from a melody, sequenced using a particular instrument’s samples. The evaluation scores we compute therefore count the number of these predictions are made correctly and incorrectly over the examined set, which contains samples belonging to multiple instruments, and these metrics can be aggregated per-instrument (e.g. “Grand Lady D”, one of the grand piano instruments drawn from the *Nord Piano Library*), or by ground-truth class (yielding per-class scores for each of the “grand” and “upright” labels).

The first result computed in order to get an overview of the number of correct and incorrect predictions made for each ground-truth label is presented in a confusion matrix. This 2-by-2 map presents the number of correctly and incorrectly labelled samples for each of the ground-truth classes. Hence, the confusion matrix entries indicate the number of true and false positives for each class as define in ***Table 6.1.1‑1***. All of the numeric scores computed on a classifier’s predictions can be derived from the confusion matrix.

|  |  |  |
| --- | --- | --- |
| **True labelPredicted** | *Grand* | *Upright* |
| ***Grand*** | True Grand | False Upright |
| ***Upright*** | False Grand | True Upright |

Table 6.1.1‑1: Outline of the confusion matrix used for evaluating the predictions of our upright/grand piano binary classifier.   
The number of correct class predictions by a system on an evaluation dataset (the number of “True Grands” and “True Uprights”) ae given by the diagonal entries (shaded in green), while the incorrect prediction counts are given by the off-diagonal entries.

Note that in the typical definition of a binary classification confusion matrix, the terms “True/False Positive” and “True/False Negative” are used. However, in our problem, the concept of “negative” and “positive” classes is a misnomer, since we value correctness and precision of predictions for either class equally in our application, regardless of their label value (0 or 1). We use these terms only to introduce the scoring metrics, in accordance with common usage.

The simplest score to compute is the accuracy, which presents the number of correctly identified samples over the total number of samples in the evaluation set. However, since this score is an unweighted average over all the evaluated samples, if the evaluation set is unbalanced it can provide misleading scores. For instance, our unseen held-out test partition features 8 individual grand piano instruments and 6 upright piano instruments. *TODO: cite the class balance percentage in one of the test sets.* Hence, a “blind” classifier which predicts only grand pianos regardless of the input data, can obtain an accuracy score of x%. Therefore evaluating the accuracy can assign high scores to classifiers with very poor ability to discriminate between the classes, and so this metric should not be applied over unbalanced evaluation datasets. Furthermore, we do not want to use metrics that favour correctness on either class, since the evaluation set imbalance is not a real-world distribution which we seek to capture in our scoring – we instead value correct classification of both classes equally.

The per-class accuracies give a more complete view of the model’s ability to discriminate between the classes, even if the evaluation set used is unbalanced. These can be calculated over each of the rows of the confusion matrix respectively, by taking the proportion of correctly identified samples relative to the total number of evaluated samples belonging to each ground-truth class. In our case, the accuracy of each class quantifies the following statements: “out of the ground-truth uprights in the evaluation support, how many samples drawn from upright pianos were predicted correctly (True Uprights); and out of the “grand” samples, how many were predicted correctly (True Grand) by the classifier?”.

From these per-class accuracies, we can calculate the balanced accuracy score, which, contrary to the standard accuracy score, aims to take class distribution into account. Balanced accuracy is computed as the mean of the accuracies achieved in each class [80], as defined in , in which abbreviations for the entries in the confusion matrix are used as follows:

* : Number of correctly predicted “upright” samples (True Upright)
* : Number of correctly predicted “grand” samples (True Grand)
* : Number of incorrectly labelled “upright” samples (False Upright)
* : Number of incorrectly labelled “grand” samples (False Grand)

The balanced accuracy is also often referred to as the macro-averaged accuracy over the classes. Computing the balanced accuracy can be seen as equivalent to manually balancing the test set by duplicating minority entries, before computing the standard accuracy.

An alternative to balanced accuracy that is often recommended as an appropriate score to evaluate unbalanced problems is the score, which is defined in (adapted from [81]).

As can be observed by comparing their definitions in and , the F1 score does not take into account True Grand counts, while balanced accuracy takes into account True Grand and True Upright counts equally. Therefore, since we do not seek to favour of one class over the other, we prefer the balanced accuracy, as this provides a general impression of the model’s discrimination ability without favouring either class. Additionally, we prefer balanced accuracy as a more interpretable metric, since it results directly from the two per-class accuracies.

Thus, in our model selection, validation and evaluation, we will consider mainly the confusion matrix, the balanced accuracy score, and the per-class accuracies achieved by the evaluated classifiers to gauge their relative performance.

### Model selection and validating the classifier

#### Training and validation loss

When comparing different architectures and hyperparameter selections for our CNN, we must perform evaluation within the development set, to prevent tailoring our final model selection to the held-out test set. Therefore, for the purposes of model selection, a validation set (part of the development partition) is used as an evaluation dataset over which to compute the metrics selected in Section 6.1.1.

Within a single run of training a model, we can analyse its training behaviour by observing the evolution of the model’s prediction loss, averaged within each epoch, over the training loop in order to observe the error reduction achieved throughout the process. Measuring this value on the training set allows us to verify that the model is able to learn from the training data throughout the loop – but a reduction of training loss is generally easily achieved for many different hyperparameter selections. The more instructive way of gauging the quality of the training process is by observing the evolution of the loss achieved on the validation set, since this gives us an idea of whether the improvements made at each epoch are generalisable to unseen data as training progresses. In our hyperparameter and model validation process, we plot both of these errors over the training loop to gauge each model’s training behaviour. The typical aspect of a plot showing both of these losses over the training loop is presented in ***Figure 5.3‑1***. This shows that a model that generalises well, without underfitting the data and overfitting to the training set, has decreasing validation as well as training loss throughout the training process.

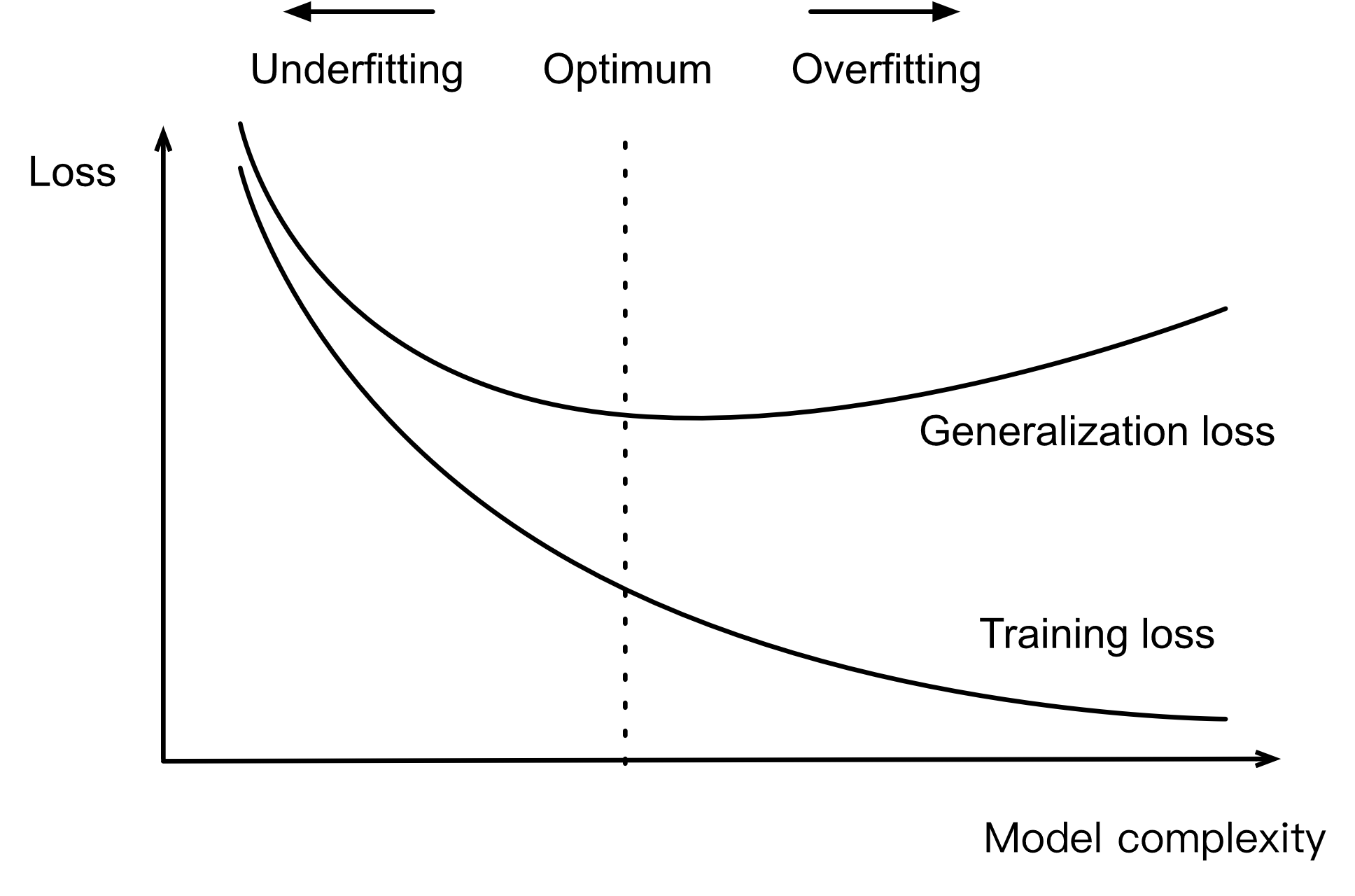


Figure 5.3‑1: Typical training set and validation set (i.e. generalisation) loss behaviour relative to model complexity. Source: [62], Figure 4.4.1.  
The x-axis is labelled as model complexity, which is the general case. In our analysis, we will substitute this for the number of epochs trained for, since this can be seen as determining the model complexity if all other hyperparameters are fixed. Indeed, the longer a model is optimised, the closer it fits the training set by increasing the learnt complexity.

#### Cross-validation

However, if we repeatedly optimise our model selection using a fixed training and validation set within the development set, we will overfit our system to perform well on the particular validation set selected, negating any generalisation ability the selected model might have on truly unseen data. Furthermore, we limit our model to only being exposed to a subset of the development set, since we cannot train the model on validation set samples. In cases such as ours, in which the dataset is fairly limited in size and additional data is difficult to acquire, we would benefit from exposing a model to all development samples during the model selection process.

To address these concerns, we perform k-fold cross validation on the development set to select optimal hyperparameters without overfitting to any one data subset. As explained in [62], cross-validation aims to estimate the expected accuracy that could be achieved on the unseen test set, by splitting the development set into K subsets (or folds), then training and validating K independent classifiers sharing the same architecture and hyperparameters. Each classifier is validated on a different subset by iterating through the combinations of train/validation partitions, using K-1 folds as training data and the remaining fold as a validation set. In order to train each fold’s model on a maximal amount of data, larger values of K are preferred to give the best approximation of unseen performance averaged over the K models. However, given the small number of unique instruments contained overall in the development set (23 different pianos in total), K=4 was selected in order to ensure a sufficient number of pianos would appear in each fold to be used as a validation set. Using partitioning by instrument (see Section 5.2) to split the development set into the 4 folds, each fold contains 5 or 6 unique instruments, roughly half of which are grands and the other half uprights, allowing for balanced validation of each trained classifier. Cross-validation overall results are averaged across the 4 folds to produce a single set of scores for the architecture/hyperparameters under examination, weighting each per-fold score by the corresponding number of samples in the validation support.

The model selection process we apply therefore consists of:

* Search for the best-performing candidate:
  + Select a candidate architecture and hyperparameters to validate.
  + 4-fold cross-validate the candidate model architecture and hyperparameters on the development set.
  + Repeat with different candidate architecture and/or hyperparameters
* Save the best-performing architecture and hyperparameters on the basis of average classification scores obtained in cross-validation.
* Retrain a new best model using the full development set as a training set
* Evaluate the re-trained model on the held-out test set.

This development workflow is visualised in ***Figure 5.3‑2***, summarising our approach to evaluating a particular selection of architecture or hyperparameters, and the evaluation of the final model selection.

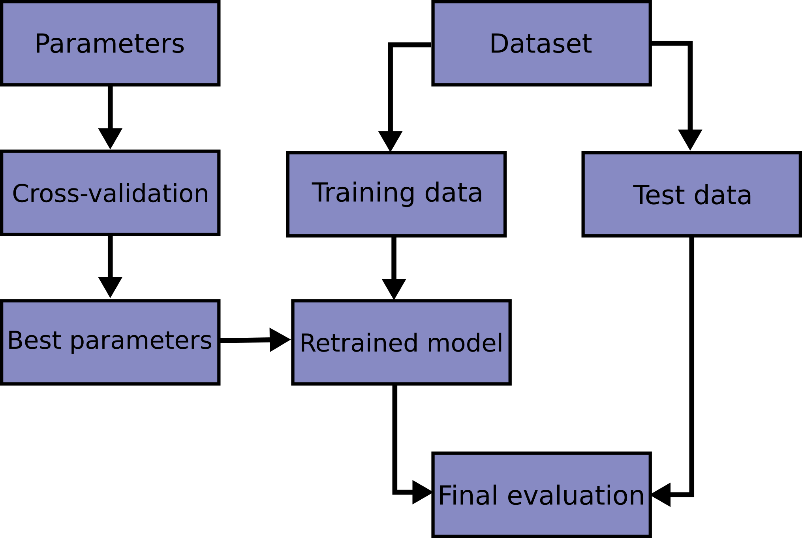


Figure 5.3‑2: Diagram summarising our model selection workflow, showing how different selections of network hyperparameters are compared using cross-validation (left-hand side) on the development set (which is made up of training data in our case), before selecting the best hyperparameters retrain a model on the full training data for evaluation on the held out test set (right-hand side). Source: [82].

#### Hyperparameter search

In order to find the hyperparameters producing the best cross-validation performance, we automate the process using an exhaustive search over a pre-defined set of candidate values for each considered parameter. In our experiments, we considered the search spaces given in ***Table 6.1.2‑1*** to tune the hyperparameters for each of the proposed systems. Each range selected for this purpose was informed by initial trial-and-error experimentation, to determine which approximate values to aim for with each parameter so as to ensure reasonable training behaviour (observed in training and validation loss plots). This search range was also constrained by the available time and computational resources, since the exhaustive parameter search applied in conjunction cross-validation involves training a large number of models for each parameter combination. Therefore, we note that the optimal combination of hyperparameters may not have been considered, as only a small subset of the total search space is considered in our experiments.

|  |  |  |
| --- | --- | --- |
| Hyperparameters | **Single note Classifier** | **Melody-based Classifier** |
| ***Epochs*** | [15, 20, 25] | [20, 25] |
| ***Learning Rate*** | [0.001, 0.002, 0.003] | [0.001, 0.002, 0.003] |
| ***Batch Size*** | [64, 128, 256] | [128, 256, 512] |

Table 6.1.2‑1: Search spaces explored in our hyperparameter searches for each tackled task (isolated-note-based classification and melody-based classification).

In order to automatically compare cross-validation performance between candidate selection, we must select an optimality criterion from the average performance scores returned by cross-validation. As described previously, the best way to get a clear impression of our classifier’s performance is to consider its accuracy on each class of piano, yielding two values. However, to reduce these values to a model selection criterion in hyperparameter search, we need to combine the two per-class values into a single, optimisable, metric. The balanced accuracy, or macro-averaged accuracy, is a good candidate, but we want to avoid large discrepancies between the per-class scores (which would exist for example for a model achieving 0.3 accuracy on one class and 0.8 on the other, giving an inflated balanced accuracy score of 0.55). Therefore, an informed choice for reducing the per-class scores to a single value is taking the smallest of the two per-class accuracies, in order to optimise both of the per-class performances jointly and prevent being misled by unbalanced performance on the two classes. We must however take care that the resultingly selected model produces better than chance performance, i.e. that the overall macro-averaged accuracy is above 0.5. Therefore, the two conditions for updating the current best hyperparameter set in our exhaustive search algorithm are that:

* Higher than the previous best minimum of per-class-accuracies is achieved, and
* The overall macro-averaged (balanced) accuracy rate is above 0.5,

both figures resulting from weighted averages over a 4-fold cross-validation, as explained previously. Therefore, only better-than-chance performance will be accepted in our model selection.

# Implementation

## Software standards and toolkits

Unless otherwise stated, all software development for the system was performed in Python, using the PyCharm Integrated Development Environment (IDE), choices made as a result of prior personal experience, as well as the popularity of Python in the machine learning community and the availability of open-source neural network development libraries. A key aspect of several of the software libraries used is the fact that they are open-source, allowing for the examination of the underlying source codebases for understanding, debugging and modification if needed.

### Signal processing feature extraction toolkits

* VOICEBOX for MATLAB [83]: includes a wide range of standard audio Digital Signal Processing functions, including timbral analysis, for voice processing, many of which are appropriate also for processing musical audio. We used VOICEBOX and MATLAB in our initial experiments on formant analysis by Linear Predictive Coding (see Section 3.1.6).
* The Librosa [84] library for Python: includes functions for MIR signal processing algorithms, including spectral, harmonic, statistical, and temporal analysis, and extraction of various timbral features. In particular, we use Librosa for the generation of Mel-spectrograms, as well as audio file resampling to convert the BiVib waveforms to the required sample rate.

### Machine learning and data science libraries for Python

* NumPy [85] is used for MATLAB-like efficient vectorised mathematical operations, matrix functions and data manipulations.
* Pandas DataFrame objects are used to manage, store and operate on the dataset.
* PyTorch [86] was used to design, train and test neural network architectures using tensor-based operations.
* PyTorch-CUDA is employed to leverage Graphical Processing Unit (GPU) hardware to increase speed of tensor operations, in conjunction with the Nvidia cuDNN toolkit.   
  The Google Colab Pro cloud computing platform is also used in order to train models faster on high-performance remote CUDA hardware optimised for deep learning, especially for the hyperparameter searches performed.
* Scikit-learn [87] is used for additional machine learning and data science functions such as the evaluation of classification scoring metrics.

### Other libraries used

* The Pickle Python library is used to save variables from memory to cold storage, so that generated and pre-processed data can be loaded without re-computation. This is particularly useful for loading our unified single-note waveform dataset without needing to read the original archives for *MAPS* and *BiVib*, since we only use a subset of each of these large data repositories in our combined dataset. Trained models are also saved using Pickle in order to re-evaluate models several times, and save progress.
* The Mido Python library was used to send MIDI messages for remotely triggering the digital piano in order to resample the Nord Piano Library. Mido was also used in the implemented sequencer, in order to decode the monophonic MIDI files used to generate melodies.

## System software structure

In this sub-section, we give an outline of the code structure of the project, detailing each of the implemented modules, their member functions, highlighting the specificities of the implementation. In order to validate Each module was developed iteratively, and tested individually whenever new functionality was added. We tracked progress for the project using GitHub for version control, regularly committing progress when new modules were developed. The developed source code, results obtained, and models trained for the project are made available on our GitHub repository.

*TODO: Detail code structure and object-oriented approach, how each component (class or function) was developed and tested individually, is implemented and fits together in the overall system:*

Instrument loader class (data\_loading.py):

* BiVib and MAPS loading and formatting functions.
* Sequencer script implemented from scratch specifically for the purpose of automated note-by-note MIDI sampling of the Nord Piano Library. Synchronous audio recording and MIDI message sending, looping through the considered note range and velocities. Describe how we sampled the Nord Piano Library by triggering midi and recording in order to automatically re-sampling the Nord’s instruments.
* Spectrogram generation pre-processing function: includes casting to float for compatibility with Librosa, padding waveforms to the same length, generation using Librosa (implementation uses Fast Fourier Transform STFT), spectrogram magnitude normalisation.

Melody loader class (melody\_loading.py): inherits from the instrument loader, since this requires calling the single-note loading functions as well as functions for spectrogram generation.

* Monophonic MIDI sequencing function implemented from scratch specifically for this purpose. Describe the melody sequencer script written for the generation of the melody dataset. Melodies were kept within the targeted set of 25 notes by ignoring MIDI files containing a larger range of notes, and transposing any MIDI files containing melodies fitting within this span but in a different range of pitches.
* Spectrogram generation pre-processing function applied to the generated melodies

CNN classes (timbre\_CNN.py): one for each CNN design (the single note classifier and the melody classifier):

* SingleNote/MelodyTimbreCNN: inherits from PyTorch’s standard NN class so that all the convenient functions and classes supplied by PyTorch for constructing and training a CNN can be used.
* SingleNote/MelodyTimbreCNNSmall: inherits from SingleNoteTimbreCNN so that inference (forward()) function and helper functions can be shared. Specifies only a different architecture by setting different CNN layers from its parent class in the constructor.

Classifier scoring functions (evaluation.py):

* Score computation over the input predictions and ground truth labels.
* Score display, printing tables presenting the scores and plotting the confusion matrix.

Run-time functions (run\_TimbreCNN.py):

* Dataset partitioning function, with the desired mode, number/size of each partition, and random seed passed in as parameters. Various dataset partitioning modes are implemented as outlined in Section 5.2, and these can be easily
* Model training function, which takes in as parameters the desired type of model with the targeted training set. This creates a model and trains it using the hyperparameters set as global variables. If specified in the parameters, this function also returns a plot of the loss curves for the training run (outlined in Section
* Model evaluation function, which takes in as parameters the already trained model under evaluation and the targeted test set. This function passes the data through the model in inference mode and computes scores by comparing network predictions to ground truth labels.
* Cross-validation function, which takes in as parameters the type of model we want to validate, the training/validation set we want to split for 4 fold cross-validation, and the desired partition mode to apply. This calls the partitioning, training and evaluation functions for each fold, then calculates the statistics of the validation scores over the folds.
* Hyperparameter search function which searches a specified search space for each considered parameter, performs.
* Main function, which loads the required pre-processed data, performs cross-validation and hyperparameter search, retrains a model, and evaluates it on the validation and test sets.

Reference my GitHub code repository for the source code of the project.

# Classification Results

In this section, we present the outcomes of the CNN model selection process described in 6.1.2, namely the hyperparameter search results for various combinations of architectures and tasks, detailing the statistics of the best scores obtained over the corresponding 4-fold cross-validations. On the strength of these results, we select the final architectures and hyperparameter sets for use in the final re-trained models, one model for each task. We re-train final models using the best performing architecture and hyperparameter combinations, and present the scores obtained by these final models on the held-out, unseen test set.

## Model selections & training behaviour for each task

The highest-performing architectures and hyperparameters were selected by their cross-validation performance, on the basis of the highest achieved minimum-per-class accuracy scores, as detailed in Section 6.1.2. We review the best-achieved cross-validation scores and analyse training behaviour for each of the two proposed architectures, applied first to the single-note classification, then to the melody-based classification problem.

### Single-note classification

The hyperparameter search results and scores for the proposed TimbreCNN and TimbreCNNSmall architectures applied to the single-note task are presented in ***Table 8.1.1‑1***. We observe that the smaller architecture (column entitled *SingleNoteTimbreCNNSmall*, key results highlighted) produces slightly higher mean scores over the 4 cross-validation folds, both in terms of the overall balanced accuracy achieved and in terms of the mean minimum-per-class accuracy. The standard deviations for each of these scores are comparable between the two architectures, as plotted in *Error! Reference source not found.* and further discussed in Section 8.2. Furthermore, while the per-class accuracy in class “grand” of the *SingleNoteTimbreCNN* architecture is higher than that achieved by the small architecture, *SingleNoteTimbreCNNSmall* is more consistent across the two targeted classes, with a smaller spread of mean scores between the classes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Architectures:* | | | ***SingleNoteTimbreCNN*** | | ***SingleNoteTimbreCNNSmall*** | |
| **Hyper-parameters** | | *Epochs* | 25 | | 20 | |
| *Learning Rate* | 0.002 | | 0.002 | |
| *Batch Size* | 128 | | 64 | |
| **Cross-validation scores** | | | **Mean** | **Standard deviation** | **Mean** | **Standard deviation** |
| ***Balanced accuracy*** | | | 0.714 | 0.039 | 0.744 | 0.047 |
|  | **Class accuracy: *Grand*** | | 0.801 | 0.047 | 0.788 | 0.049 |
|  | **Class accuracy: *Upright*** | | 0.627 | 0.101 | 0.700 | 0.106 |
| ***Min.-per-class accuracy*** | | | 0.627 | 0.101 | 0.692 | 0.096 |

Table 8.1.1‑1: 4-fold cross-validation statistics (lower-half) obtained by the optimal selected hyperparameters for each of the two proposed architectures (upper-half), applied to the single-note classification problem.

We note that the mean minimum-per-class accuracy over the cross-validation is not always equal to the minimum of the mean class-accuracies (as is the case for SingleNoteTimbreCNNSmall), since the class with the smaller accuracy is not always the same from one cross-validation fold to the next. For either of the considered architectures, the mean class accuracy achieved is lower in the “upright” class than the “grand” class - both architectures are more likely to mislabel upright piano samples as belonging to the “grand” class than the other way around. This implies that both model types tend to default to predicting the “grand” class, which may be a reflection of the slight class imbalance present in the development set towards the “grand” class.

While slight, the score improvement achieved by the smaller architecture is particularly noteworthy since this architecture has 4.6 times fewer learnable parameters than the standard TimbreCNN architecture, and despite this, achieves comparable classification performance. This implies that the simpler TimbreCNNSmall architecture may be better-adapted to the single-note piano classification problem, and that the additional complexity afforded by the larger TimbreCNN architecture does not seem to help in this approach of the piano classification problem. This may be evidence of model over-parametrisation for the latter architecture, given the relatively simple nature of the single-note piano classification task (at least when compared to the more complex inputs presented by the melody-based piano classification task).

Comparing the hyperparameters returned by the local search performed for each architecture, we see in the upper portion of ***Table 8.1.1‑1*** that the optimal SingleNoteTimbreCNNSmall architecture requires fewer training iterations and smaller batch sizes than SingleNoteTimbreCNN, which again indicates that the advantage gained by the higher-performing architecture is afforded to it by model simplicity. Specifically, the 5 additional iterations required to train the SingleNoteTimbreCNN likely correspond to the additional steps required to fit the larger number of parameters to the training data, where each step is informed by twice as many data samples than the smaller architecture.

As shown in ***Figure 8.1‑1***, this additional model complexity of SingleNoteTimbreCNN compared to the smaller architecture results in a wider generalisation gap between the training and validation set errors as training progresses. Therefore, it seems that the SingleNoteTimbreCNN architecture, with the selected hyperparameters, suffers from overfitting, leading to poor generalisation performance. Conversely, the more compact SingleNoteTimbreCNNSmall architecture and optimal hyperparameters exhibits more favourable training behaviour in ***Figure 8.1‑2***, since its validation loss decreases continuously throughout the 20 epochs of training. However, the training loss does not behave as smoothly, most likely as a result of the smaller batch size of 64 compared to the more complex model’s 128; as detailed in Section 5.3.2, smaller batch sizes can cause noisy optimisation behaviour which can lead to inconsistent convergence from one training session to the next.

|  |  |
| --- | --- |
| Figure 8.1‑1: Loss curves over a training run using optimal SingleNoteTimbreCNN hyperparameters. (model name: SingleNoteTimbreCNN - model\_128\_25\_0.002\_plot) | Figure 8.1‑2: Loss curves over a training run using optimal SingleNoteTimbreCNNSmall parameters. (model name: SingleNoteTimbreCNNSmall - model\_64\_20\_0.002\_plot) |

*In the above figures, the training (red line) and validation (blue line) set losses result from training models using the optimal hyperparameters for each architecture. A fixed balanced train/validation split is employed within the development set, partitioned by instrument so as to reproduce a typical cross-validation fold.*

Therefore, for the single-note piano classification task, we select SingleNoteTimbreCNNSmall as the optimal architecture, paired with the following training hyperparameters:

* Batch size: 64
* Epochs: 20
* Learning rate: 0.002

### Melody-based classification

We now turn our attention to the application of each of the two proposed architectures (the initial TimbreCNN and the more compact TimbreCNNSmall) to the melody-based classification task. While the input features are of the same dimensions and type as in the single-note case, the input spectrograms for the melody-based task contain richer information, each spectrogram segment being drawn from a melody and featuring multiple different notes played monophonically on a particular piano instrument. Therefore, we expect different architectures and hyperparameter selections to be suited to this more complex timbral analysis task, as well as an increase in classification performance, since a higher quality and quantity of timbrally-relevant context is contained in each melody dataset sample compared to the isolated-notes data supplied to single-note classifiers.

***Table 8.1.2‑1*** presents the hyperparameter search results and best cross-validation performance of each architecture applied to this task. Comparing the mean scores achieved by the two architectures, we see that the larger MelodyTimbreCNN architecture obtains slightly higher scores compared to MelodyTimbreCNNSmall, both in terms of overall balanced accuracy and mean minimum-per-class accuracy (key results highlighted). The per-class results show more consistent performance for MelodyTimbreCNN, with a smaller discrepancy between average accuracy scores obtained in each class. Both architectures achieve sizable classification performance gains compared to the results of the single-note systems seen in ***Table 8.1.1‑1***. However, the standard deviations of the considered scores are relatively large for both architectures; this variability is further discussed in Section 8.2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Architectures:* | | | ***MelodyTimbreCNN*** | | ***MelodyTimbreCNNSmall*** | |
| **Hyper-parameters** | | *Epochs* | 25 | | 20 | |
| *Learning Rate* | 0.003 | | 0.003 | |
| *Batch Size* | 128 | | 512 | |
| **Cross-validation scores** | | | **Mean** | **Standard deviation** | **Mean** | **Standard deviation** |
| ***Balanced accuracy*** | | | 0.827 | 0.094 | 0.814 | 0.078 |
|  | **Class accuracy:** *Grand* | | 0.847 | 0.061 | 0.858 | 0.124 |
|  | **Class accuracy:** *Upright* | | 0.806 | 0.180 | 0.770 | 0.084 |
| ***Min.-per-class accuracy*** | | | 0.744 | 0.144 | 0.742 | 0.102 |

Table 8.1.2‑1: 4-fold cross-validation statistics (lower-half) obtained by the optimal selected hyperparameters for each of the two proposed architectures (upper-half), applied to the melody-based classification problem.

As opposed to the single-note task, the more complex model achieves higher performance – this suggests that the additional complexity of the larger MelodyTimbreCNN is useful to leverage the richer inputs for this task, as well as the larger pool of training data available in the melody dataset. Comparing the locally optimal hyperparameters found for each architecture, we see that MelodyTimbreCNN requires 5 additional training iterations, but a much smaller batch size, than the smaller architecture. The additional training time again likely accounts for the greater number of parameters to learn in the larger architecture. We also note that for both architectures, the batch sizes selected for this task are larger than those selected for the single-note task, in accordance with the larger training set available in the melody dataset compared to the single-note data (refer to sections 4.3.2 and 5.1.4 to understand why there is a difference in dataset size between the single-note and melody-based tasks).

The training behaviours exhibited in ***Figure 8.1‑3*** and ***Figure 8.1‑4*** shows reduction of the validation set loss over the training loops for both architectures using the optimal hyperparameters, suggesting reasonable generalisation ability for both models. However, both the training and validation curves for the larger MelodyTimbreCNN (***Figure 8.1‑3***) show a degree of volatility as opposed to the smoother loss evolutions observed for the smaller architecture (***Figure 8.1‑4***). This noisier convergence in the former case likely results from the smaller batch size of 128 used, as opposed to the 512 samples per batch used in training of MelodyTimbreCNNSmall, since the batch size dictates the stability of gradient descent optimisation.

|  |  |
| --- | --- |
| Figure 8.1‑3: Loss curves over a training run using optimal MelodyTimbreCNN hyperparameters. (model name: MelodyTimbreCNN - model\_128\_25\_0.003\_plot) | Figure 8.1‑4: Loss curves over a training run using optimal MelodyTimbreCNNSmall hyperparameters (model name: MelodyTimbreCNNSmall - model\_512\_20\_0.003\_plot) |

*In the above figures, the training loss is represented by a red line, and the validation set loss is shown as a blue line. The same methodology is applied here as the generation of Figure 8.1‑1 and Figure 8.1‑2*.

Given its higher overall cross-validation scores compared to the smaller architecture, we therefore select MelodyTimbreCNN as the optimal architecture for the melody-based classification task, using the following training hyperparameter selection:

* Batch size: 64
* Epochs: 20
* Learning rate: 0.002

We note however the wide generalisation gap observed between training and validation losses in the training behaviour of this architecture (***Figure 8.1‑3***), which suggests this type of model may be prone to overfitting.

## Comparison of cross-validation results across both tasks

To compare the optimal performance of each of the proposed architectures across the two considered tasks, we plot in *Error! Reference source not found.* the statistics of the cross-validation results returned by the hyperparameter searches for each of the two proposed CNN architectures (TimbreCNN and TimbreCNNSmall), applied to both of the single-note and melody-based tasks.

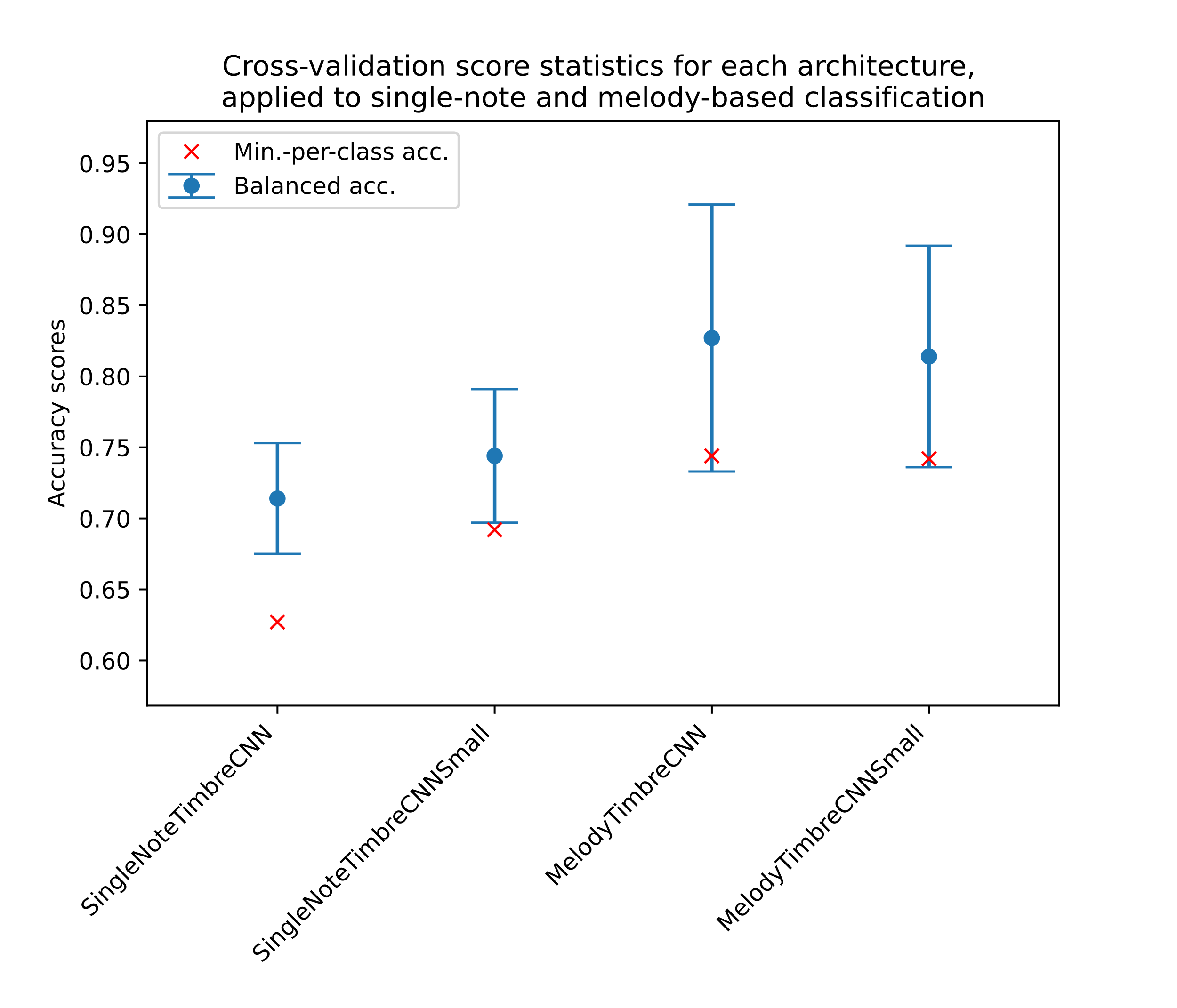


Figure 8.2‑1: Cross-validation scores obtained from the hyperparameter search results for both architectures applied to each problem.

* Blue points and error bars represent the mean and standard deviations of the balanced accuracy scores respectively, computed over the 4 cross-validation folds.
* Red crosses indicate the mean over the 4 folds of the minimum-per-class accuracies achieved. For these scores, no error bars are used, since their standard deviation may not represent the spread of classification performances across the different models trained, since the minimum-accuracy class can be different in each fold.

Across all four types of model examined in ***Figure 8.2‑1***, we see that the MelodyTimbreCNN achieves the highest mean balanced accuracy score as well as the highest mean minimum-per-class accuracy score. Comparing the single-note classifiers to the melody-based classifiers, we see a notable improvement in cross-validation performance achieved in the latter formulation of the problem. Indeed, the scores achieved by the single-note systems are much closer to the baseline chance-accuracy score of 0.5, indicating that this formulation makes for less reliable classification of piano types. This is expected, since the two melody-based classifiers make predictions based on richer information with more musical context, namely by analysing more than one isolated note and pitch at a time.

However, we observe from the error bars in ***Figure 8.2‑1*** that the balanced accuracy scores achieved are much less consistent for the melody-based classifiers than for the single-note systems, with a large fluctuation between individual scores attained by the models trained in each of the 4 cross-validation folds in the melody-based systems. These large standard deviations, especially for the MelodyTimbreCNN architecture and hyperparameter selection, reveal inconsistent performance from one fold to another. This may be a sign that these models are susceptible to overfitting to the training set they are given, since an independent training set is supplied to the model trained in each fold. This hypothesis is consistent with the observations made on the training behaviour of these models in Section 8.1.2.

We note that due to the difference in overall dataset size (i.e. the number of samples available for model selection and training) between the single-note and melody-based formulations, the results listed for the single-note systems are less statistically significant than those listed for the melody-based classifiers, since the validation sets used to calculate the scores in the single-note case contain fewer samples. Nevertheless, this larger number of samples in the melody dataset results only from the data-augmentation we applied (see Section 4.3.2) to generate it. Therefore, we still consider it pertinent to compare the two approaches, since the datasets for each task are both derived from the same raw single-note recordings drawn from the same set of individual piano instruments.

## Held-out test set results

Given the architecture and hyperparameter selections made in Section 8.1 for each task, we seek to evaluate them on the unseen test set. Namely, we evaluate the SingleNoteTimbreCNNSmall architecture using the hyperparameters listed in ***Table 8.1.1‑1*** for single-note classification, and the MelodyTimbreCNN architecture with the hyperparameters listed in ***Table 8.1.2‑1*** for melody-based classification. We retrain new models for each of these, this time using the full development set for training, as we no longer require a separate validation set. These final re-trained models are then evaluated using the held-out test set, to gauge the models’ abilities to generalise to unseen data and simulate “real-world” performance.

### Overall performance

The test set confusion matrices and classification scores achieved by the two considered re-trained models, SingleNoteTimbreCNNSmall and MelodyTimbreCNN, are presented in ***Figure 8.3‑1*** and ***Table 8.3.1‑1***. While the overall balanced accuracies between the two systems are in a close neighbourhood of one another, the per-class accuracy scores and the confusion matrix show that the melody-based system made higher quality predictions especially when identifying ground-truth grand piano samples.

In particular, the confusion matrix for the single-note system (***Figure 8.3‑1*** – Left) shows many ground-truth “grand” samples being labelled incorrectly as “upright”, while few mistakes are made on the “upright” class. The prevalence of “false uprights” indicates that this classifier is biased toward predicting “upright” for most sample. As a result, the minimum-per-class accuracy of 54%, achieved over the “grand” class (see ***Table 8.3.1‑1*** - *SingleNoteTimbreCNNSmall* column, key result highlighted), is very close to a chance-prediction baseline of 50%. Therefore, the trained SingleNoteTimbreCNNSmall model is not able to generalise well to unseen data.

On the other hand, the confusion matrix for the melody-based classifier (***Figure 8.3‑1*** – Right) has a majority of the test-set samples falling into the diagonal entries (the true positives for each class), demonstrating a much finer ability to discriminate between the two classes in its predictions. Furthermore, as shown in the *MelodyTimbreCNN* column of ***Table 8.3.1‑1***, the melody-based classifier fairly unbiased, achieving comparable per-class accuracies of approximately 79% for grand samples, and 78% for upright samples.

|  |  |
| --- | --- |
|  |  |

Figure 8.3‑1: Confusion matrices of the test-set predictions produced by the retrained models.  
Left: Re-trained architecture & hyperparameters selected for single-note classification.  
Right: Re-trained architecture & hyperparameters selected for melody-based classification.  
Shading intensity indicates the proportion of the total test set’s samples falling into each entry. Refer to Table 6.1.1‑1 as a guide for reading confusion matrices.

|  |  |  |  |
| --- | --- | --- | --- |
| **Scores** **Models** | | ***SingleNoteTimbreCNNSmall*** | ***MelodyTimbreCNN*** |
| **Balanced Accuracy** | | 0.761 | 0.783 |
|  | **Class Accuracy**: *Grand* | 0.540 | 0.790 |
|  | **Class Accuracy**: *Upright* | 0.982 | 0.775 |

Table 8.3.1‑1: Classification scores obtained per-class and macro-averaged on the test set, for the models re-trained for each task. Key results showing the higher-performance of the MelodyTimbreCNN are highlighted in green and red.

Therefore, we find that the melody-based classifier has learnt generalisable embeddings, since it is able to achieve comparable results in held-out testing to those obtained in validation, as opposed to the single-note classifier, which shows poorer classification performance on unseen test examples.

### Per-instrument results

*TODO: Per-instrument results*

* Table with per-instrument scores
* Maybe plot the per instrument scores for both systems on one figure, so that we see the outliers. Instrument labels on the x axis and prediction accuracy on the y-axis.

# Conclusions, Discussion and Further work

To conclude, we summarise the key achievements of the project, namely:

* The design and selection of CNN models (see Section 5.3) tailored to the task of classifying the timbre of upright and grand piano sounds from spectrogram inputs, achieving promising classification performance (see Section 8).
* The design of a suitable pre-processing pipeline for spectrogram generation (see Section 5.1), informed by our review of the background theory on perceptual feature-sets (Section 3.1.7).
* The assembly of a uniformly-formatted dataset of piano sounds from multiple sources targeting the considered machine learning problem, as well as the introduction of a novel approach to musically-motivated data-augmentation for the melody-based system (see Section 4.3).

In the following discussion, we evaluate the results obtained in Section 8 in order to address the key research questions posed at the inception of the project, before going over the limitations of our work as well as the next steps which could be taken to deepen our approach.

## Evaluation

Overall, given the analysis and results presented in Section 8, we have found promising generalisation performance when using the MelodyTimbreCNN in the melody-based problem. The performance achieved by this system on unseen data is markedly better than chance. While the absolute accuracy scores achieved of approximately 78% are still far from perfect prediction, we note that the chosen task of discriminating the sound of upright and grand pianos can be difficult for the human untrained ear. Therefore, we can draw the conclusion from this promising result that the chosen approach of spectrogram-based CNN inference is a viable way of performing musical timbre analysis tasks such as piano-type classification, which involve analysing the finer timbral differences that exist between the sounds produced by closely related instruments. Therefore, our principal objective for the project of developing a generalisable machine learning method for piano identification by timbre have been addressed by the proposed system, although it is difficult to provide a basis of comparison to quantify the extent to which our system “reliably” identifies timbre, and how this compares to the human ability to do so.

Our results also show that, in general, melody-based classifiers demonstrate superior piano classification performance to that achieved using the single-note systems. This is an expected result, since the melody-based task was designed to pre-condense more musical information and context into each input sample for a CNN to extract. Namely, the melody-based classifiers are able to make predictions on the strength of a more complete snapshot of a particular piano’s sound in the form of a melody spectrogram, as opposed to a single-note spectrogram. Additionally, the melody-based models benefit from a larger pool of training material, since the melody-generation process augments the single-note dataset. The resulting performance gains show that the proposed architectures are able to draw the benefit of using this richer musical context. Thus, from this result we can claim that melody-based classification is a more adapted formulation of the piano timbre identification problem than single-note-based analysis. This also gives us additional insight into the viability of our spectrogram-CNN approach for timbral analysis, since performance seems to scale with the amount of musically-relevant information supplied to the models.

The smaller, more refined, architecture (TimbreCNNSmall), which we tailored to the musical domain by attempting to structurally integrate timbral concepts such as pitch invariance (as detailed in Section 5.3.1), shows promise for application to both the single-note and melody-based tasks, as evidenced by the cross-validation results (***Figure 8.2‑1***). However, the best selected model for the melody-based task, which was also the best-performing model overall in our experiments, does not utilise this architecture. This was surprising, given the performance gains achieved using this architecture for the single-note problem. However, perhaps a similarly architecture utilising musical-domain intuition with more learnable parameters is required to analyse the more complex input spectrograms used as input in the melody-based task. To demonstrate this, we would need to trial a third architecture design, retaining the narrower structure and timbrally-motivated aspects of the small architecture, but with more convolutional filters or layers – making for a CNN more closely aligned with the “deep and narrow” paradigm in deep learning applied to perceptual tasks.

## Limitations

Several potential caveats apply to the results obtained. Firstly, only one re-trained model is used to evaluate the final architecture and hyperparameter selections on the unseen test set in Section 8.3. This results in part from an effort to avoid selecting models based on unseen test set results, since this would constitute fitting the models to the held-out data, defeating the purpose of the unseen test set. We instead informed our model selection using the development set exclusively, however the large standard deviations of the scores obtained in cross-validation, especially for the melody-based models (see ***Figure 8.2‑1***), reveals a degree of inconsistency from one training run to the next, for the same architecture and hyperparameters. This raises the question of whether we got lucky to train a MelodyTimbreCNN model that happens to perform particularly well on the unseen test examples, and how this model compares to the pool of potential models that could be generated using the given architecture, hyperparameters and training set.

Secondly, we cannot claim to have found generally optimal hyperparameters for each architecture using our hyperparameter search, since we only explored a limited local search space as defined in ***Table 6.1.2‑1***. We therefore accept that other hyperparameter combinations may provide better classification performance for each of the proposed architectures. Another possible drawback of our hyperparameter search process is the risk of overfitting to the development set, as a result of repeated cross-validation on the same set of development examples available to us.

Beyond model selection, a principal limitation of the performed experiments is that the data pre-processing parameters were set heuristically, intuitively, or based on considerations drawn from the literature. This resulted from the fact that many the pre-processing steps applied such as spectrogram generation, involve a large number of tuneable parameters. A thorough experimental evaluation of their individual influences on the overall system’s classification performance was beyond the scope of our experiments, since this would have involved computing the entire data pipeline for feature extraction for every combination of parameters, in addition to the model selection experiments we performed. We therefore accept that more experimentally-informed pre-processing methods may have been applicable, and that a deeper exploration of their parameters may produce a more carefully-crafted timbral analysis system with more significant results.

We also note the lack of bases of comparison for our system’s results, since the scope of our work was more narrowly-focused than related work on timbral differentiation within a type of instrument. Therefore, we did not attempt to apply our system to related problems, which may have allowed us to compare performance with the musical instrument classification systems presented in our Literature Review (see Section 2).

*What are the advantages, disadvantages of your approach compared with related work? How does the scope of your work differ from related work?*

## Potential for Further Work

* Tailoring an architecture to the melody-based task, based on the TimbreCNNSmall but deeper in order to account for the complexity of melody-based classification.
* To mitigate the large variances in cross validation scores evidencing inconsistent performance from one trained model to the next – would benefit from additional regularisation in order to stabilise the training process and prevent overfitting to the training set. Stabilising training using regularisation methods such as dropout and early stopping.

Evaluating the classifier’s generalisation and interpretability:

* Sampling/recording a piano from a new source (either from an unused dataset or in the field) in order to evaluate the final system’s ability to generalise to unseen data recorded in different conditions from any of the used train/validation/test sets.
* Evaluating how & why the timbral classification works: plot filters/activations of different layers in the CNN in order to interpret the results in terms of the type of timbral information extracted by the network at different levels of abstraction.
* Evaluating the amount of training data used
* Interpretability of the networks

*TODO: describe how the size of the dataset was evaluated*, using learning curves: Dataset size evaluation by plotting the prediction accuracy of models trained on subsets of various sizes of the actual training data, in order to determine whether the amount of data used is a limiting factor in the performance of the classifier. For instance, if we find that the performance gains brought by increasing the portion of training data used taper off as we approach full training set utilisation, we can rule out the hypothesis of the dataset being too small for a given architecture.

# Appendix A – Review of isolated-note datasets

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Dataset name** | **Authors / Reference** | **Intended purpose** | **No. / type of instruments** | **Timbral Annotations** | **No. / type of pianos** | **No. of samples** | **Availability** |
| *BiVib* | Papetti et al. [69] | Research | 2  Pianos | Type & Model  Dynamics | 2  Grand, Upright | 1000+ | Free online |
| *Concert Piano* | N. Plath [88] | Research | 1  Piano | Model  Dynamics  Conditions | 1  Grand, before and after concert use | 600+ | Free online |
| *conTimbre* | T. Hummel [89] | Various | 150  Orchestra | Articulation | 1  N/A | 4073 | Paid online |
| *Living Room Upright* | Keypleezer [90] | Music Creation | 1  Piano | Dynamics  Articulation (pedal) | 1  Upright | N/A | Free online |
| *MAPS* | Telecom ParisTech [70] | Research | 9  Pianos | Articulation  Type & Model  Dynamics  Conditions | 9  Grand, Upright, Hybrid | N/A | Free online |
| *MIS* | University of Iowa [91] | Research | 30+  Orchestra | Model  Dynamics | 1  Grand | N/A (large) | Free online |
| *MUMS Revised* | Eerola et al. [92] | Research | 100+  varied | Model Articulation | 3 Upright, Grand | N/A (large) | No longer available |
| *Nord Piano Library* | Clavia Nord AB [71] | Music Creation | 24  Pianos | Dynamics  Articulation (pedal) | 24  12 Upright  12 Grand | N/A (large) | Included with Nord digital pianos |
| *NSynth* | Google AI Magenta [93] | Research | Multiple | Articulation  Type  Dynamics | 4  Acoustic, Electric | 305979 | Free online |
| *Piano in 162* | Ivy Audio [94] | Music Creation | 1  Piano | Dynamics  Articulation (pedal) | 1  Grand  Steinway model B | N/A | Free online |
| *Piano Pedalling* | L. Beici [95] | Research | 1  Piano | Articulation (pedal)  Type & Model  Dynamics | 1  Grand | 500+ | Free online |
| *Pianobook* | C. Henson [96] | Music Creation | 450+  varied | Articulation (pedals)  Type/Model  Dynamics | 100+ Grand, Upright, Electric | N/A (large) | Free online |
| *RWC* | Real World Computing Partnership [97] | Research | 50  varied | Articulation  Dynamics | 5  Acoustic, Electric | 2000+ | Unavailable |
| *SHARC* | G. Sandell [98]  Derived from MUMS, only steady-state portions | Research | 39 orchestra | Articulation | None | 1338 | Free online |
| *SOL* | IRCAM [99] | Research | 16  wind + string | Articulation | None | 25000 | Free online |

Table 8.3.2‑1: Review of studied single-note musical instrument sample databases, including the type of instruments sampled and online availability

# References

|  |  |
| --- | --- |
| [1] | N. M. McLachlan, “Timbre, Pitch, and Music,” Oxford Handbooks, 2016. [Online]. Available: https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199935345.001.0001/oxfordhb-9780199935345-e-44. [Accessed January 2021]. |
| [2] | I. Fujinaga, “Machine Recognition,” in *Proceedings of International Computer Music Conference*, Ann Arbor, MI, 1998. |
| [3] | G. Agostini, M. Longari and E. Pollastri, “Musical Instrument Timbres Classification with Spectral Features,” *EURASIP Journal on Advances in Signal Processing,* 2003. |
| [4] | D. Fourer, J. L. Rouas, P. Hanna and M. Robine, “Automatic Timbre Classification of Ethnomusicological Audio Recordings,” in *International Society for Music Information Retrieval Conference (ISMIR 2014)*, Taipei, Taiwan, 2014. |
| [5] | J. C. Brown, O. Houix and S. McAdams, “Feature dependence in the automatic identification of musical woodwind instruments,” *The Journal of the Acoustical Society of Americ,* vol. 109, no. 3, pp. 1064-1072, 2000. |
| [6] | B. L. Sturm, M. Morvidone and L. Daudet, “Musical instrument identification using multiscale mel-frequency cepstral coefficients,” in *18th European Signal Processing Conference, IEEE*, Aalborg, Denmark, 2010. |
| [7] | C. Joder, S. Essid and G. Richard, “Temporal Integration for Audio Classification With Application to Musical Instrument Classification,” *IEEE Transactions on Audio, Speech, and Language Processing,* vol. 17, pp. 174-186, 2009. |
| [8] | I. Ali-MacLachlan, “Computational analysis of style in Irish traditional flute playing,” PhD thesis, Birmingham City University, Birmingham, UK, 2019. |
| [9] | F. Setragno, M. Zanoni, A. Sarti and F. Antonacci, “Feature-based Characterization of Violin Timbre,” in *25th European Signal Processing Conference (EUSIPCO)*, Kos, Greece, 2017. |
| [10] | S. J. Russell and P. Norvig, “19. Learning in Neural and Belief Networks,” in *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, NJ, Prentice-Hall, 1995, pp. 567-580. |
| [11] | P. Herrera-Boyer, G. Peeters and S. Dubnov, “Automatic Classification of Musical Instrument Sounds,” *Journal of New Music Research,* vol. 32, no. 1, pp. 3-21, 2003. |
| [12] | Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998. |
| [13] | A. Krizhevsky, I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks.,” in *Advances in neural information processing systems*, 2012. |
| [14] | Stanford Vision Lab, Stanford University, Princeton University , “Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) Results,” ImageNet, 2012. [Online]. Available: https://image-net.org/challenges/LSVRC/2012/results.html. [Accessed June 2021]. |
| [15] | B. Kainz, *COMP70010 Deep Learning Class Lecture: "AlexNet",* Department of Computing, Imperial College London, 2021. |
| [16] | K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *International Conference on Learning Representations*, San Diego, CA, 2015. |
| [17] | A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” *Artificial Intelligence Review,* vol. 53, no. 8, pp. 5455-5516, 2020. |
| [18] | K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. |
| [19] | B. Kainz, *COMP70010 Deep Learning Class Lecture: "ResNet",* Department of Computing, Imperial College London, 2021. |
| [20] | Y. Han, J. Kim and K. Lee, “Deep convolutional neural networks for predominant instrument recognition in polyphonic music,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* vol. 25, no. 1, p. 208–221, 2017. |
| [21] | J. Gómez, J. Abeßer and E. Cano, “Jazz solo instrument classiﬁcation with convolutional neural networks, source separation, and transfer learning,” in *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, Paris, France, 2018. |
| [22] | J. Pons, O. Slizovskaia, R. Gong, E. Gomez and X. Serra, “Timbre Analysis of Music Audio Signals with Convolutional Neural Networks,” in *25th European Signal Processing Conference (EUSIPCO)*, Kos, Greece, 2017. |
| [23] | M. Taenzer, J. Abeßer, S. I. Mimilakis, C. Weiß, M. Müller and H. Lukashevich, “Investigating CNN-Based Instrument Family Recognition for Western Classical Music Recordings,” in *Proceedings of the 20th International Societyfor Music Information Retrieval Conference (ISMIR)*, Delft, The Netherlands, 2019. |
| [24] | K. Choi, G. Fazekas, K. Cho and M. Sandler, “A tutorial on deep learning for music information retrieval,” 2017. |
| [25] | G. Tzanetakis and P. Cook, “Musical genre classification of audio signals,” *IEEE Transactions on Speech and Audio Processing,* vol. 10, no. 5, pp. 293-302, 2002. |
| [26] | Y. Lin, W.-C. Chang, T.-M. Wang, A. W. Su and W.-H. Liao, “Timbre-constrained Recursive Time-Varying Analysis for Musical Note Separation,” in *Proc. of the 16th Int. Conference on Digital Audio Effects (DAFx-13)*, Maynooth, Ireland, 2013. |
| [27] | F. Stoter, S. Bayer and B. Edler, “Unison Source Separation,” in *Proc. of the 17th Int. Conference on Digital Audio Effects (DAFx-14)*, Erlangen, Germany, 2014. |
| [28] | P. Esling, A. Chemla-Romeu-Santos and A. Bitton, “Generative timbre spaces with variational audio synthesis,” *CoRR,* vol. abs/1805.08501, 2018. |
| [29] | A. Bitton, P. Esling and T. Harada, “Vector-Quantized Timbre Representation,” 2020. |
| [30] | Y. Lyu, C. Liu, H. Tan, R. Xie, K. Tang and Z. Gu, “Convolutional Neural Network based Timbre Classification,” in *CIAT 2020: Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies*, Held online, 2020. |
| [31] | J. Engel, L. Hantrakul, C. Gu and A. Roberts, “ DDSP: Differentiable digital signal processing,” *arXiv preprint,* vol. arXiv:2001.04643, 2020. |
| [32] | G. Peeters, B. Giordano, P. Susini, N. Misdariis and S. McAdams, “The Timbre Toolbox: Extracting audio descriptors from musical signals,” *The Journal of the Acoustical Society of America,* vol. 130, pp. 2902-2916, 2011. |
| [33] | X. Zhang and Z. W. Ras, “Analysis of Sound Features for Music Timbre Recognition,” in *International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, Seoul, 2007. |
| [34] | ISO/IEC, *MPEG-7: Information Technology – Multimedia Content Description Interface - Part 4: Audio (ISO/IEC FDIS 15938-4:2002),* 2002. |
| [35] | P. Naylor, *Speech Processing Class Lecture: "Time-frequency Analysis",* Department of Electrical and Electronic Engineering, Imperial College London, 2021. |
| [36] | J. Deller, J. Proakis and J. Hansen, “Short-Term Processing of Speech,” in *Discrete-Time Processing of Speech Signals*, Macmillan, 1993, pp. 225-265. |
| [37] | P. Cheung, *Signals, Systems and Control Class Lecture 5: DFT & Windowing,* Department of Electrical and Electronic Engineering, Imperial College London, 2020. |
| [38] | R. Plomp and J. M. Steeneken, “Effect of Phase on the Timbre of Complex Tones,” *The Journal of the Acoustical Society of America,* vol. 46, pp. 409-421, 1969. |
| [39] | M. Müller, “Fundamentals of Music Processing: Timbre,” International Audio Laboratories Erlangen, 2015. [Online]. Available: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3\_Timbre.html. [Accessed January 2021]. |
| [40] | C. Elliot, “Attacks and Releases as Factors in Instrument Identification,” *Journal of Research in Music Education,* vol. 23, no. 1, pp. 35-40, 1975. |
| [41] | D. Shwarz, “Spectral Envelopes in Sound, Chapter 3.3,” Institut fur Informatik, Universitat Stuttgart, Stuttgart, 1998. |
| [42] | B. Atal and S. Hanauer, “Speech analysis and synthesis by linear prediction of the speech wave,” *The journal of the acoustical society of America,* vol. 50, no. 2B , pp. 637-655, 1971. |
| [43] | F. Nolan and C. Grigoras, “A case for formant analysis in forensic speaker identification,” *International Journal of Speech, Language and the Law,* vol. 12, no. 2, pp. 143-173, 2005. |
| [44] | J. McCarty, “Timbral Analysis: Formant Analysis,” CCRMA Stanford Center for Computer Research in Music and Acoustics, 2003. [Online]. Available: https://ccrma.stanford.edu/~jmccarty/formant.htm. [Accessed January 2021]. |
| [45] | D. Ellis, “Linear Prediction (LPC),” Dept. Electrical Engineering, Columbia University, 2013. [Online]. Available: https://www.ee.columbia.edu/~dpwe/e4896/lectures/E4896-L06.pdf. [Accessed January 2021]. |
| [46] | J. Smith, “Physical Audio Signal Processing for Virtual Musical Instruments and Audio Effects: Inverse Filtering,” Center for Computer Research in Music and Acoustics, Stanford, 2010. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/Inverse\_Filtering.html. [Accessed January 2021]. |
| [47] | A. Oppenheim and R. Shafer, “From Frequency to Quefrency: A History of the Cepstrum,” *IEEE Signal Processing Magazine,* vol. 21, no. 5, pp. 95-106, 2004. |
| [48] | P. Naylor, *Speech Processing Class Lecture: Speech Recognition (Part 5 - Preprocessing),* Department of Electrical and Electronic Engineering, Imperial College London, 2021. |
| [49] | Wikipedia, The Free Encyclopedia, “Piano key frequencies,” [Online]. Available: https://en.wikipedia.org/wiki/Piano\_key\_frequencies. [Accessed June 2021]. |
| [50] | J. Pons, O. Slizovskaia, R. Gong, E. Gómez and X. Serra, “Timbre Analysis of Music Audio Signals with Convolutional Neural Networks,” Music Technology Group, Universitat Pompeu Fabra, Barcelona, 2017. |
| [51] | E. Keogh and A. Mueen, “Curse of Dimensionality,” in *Encyclopedia of Machine Learning and Data Mining*, Boston, MA, Springer US, 2017, pp. 314-315. |
| [52] | A. Eigenfeldt and P. Pasquier, “Real-time timbral organisation: Selecting samples based upon similarity,” *Organised Sound,* vol. 15, no. 2, pp. 159-66, 2010. |
| [53] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “4.1. Multilayer Perceptrons,” in *Dive into Deep Learning*, 2020. |
| [54] | T. Stathaki, *Digital Image Processing Class Lecture: "Spatial Filters in Image Processing",* Department of Electrical and Electronic Engineering, Imperial College London, 2021. |
| [55] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “6.2.5. Cross-Correlation and Convolution,” in *Dive into Deep Learning*, 2020. |
| [56] | PyTorch, “PyTorch 1.8.1 documentation: Conv2d,” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html. [Accessed 2021]. |
| [57] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “6.6. Convolutional Neural Networks (LeNet),” in *Dive into Deep Learning*, 2020. |
| [58] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “6.5. Pooling,” in *Dive into Deep Learning*, 2020. |
| [59] | T. Park and T. Lee, “Musical instrument sound classification with deep convolutional neural network using feature fusion approach,” *arXiv eprint,* vol. arXiv:1512.07370, 2015. |
| [60] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “3.4.6.1. Log-Likelihood,” in *Dive into Deep Learning*, 2020. |
| [61] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “11.5. Minibatch Stochastic Gradient Descent,” in *Dive into Deep Learning*, 2020. |
| [62] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “4.4. Model Selection, Underfitting, and Overfitting,” in *Dive into Deep Learning*, 2020. |
| [63] | S. J. Pan and Q. Yang, “A Survey on Transfer Learning,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 22, no. 10, p. 2010, 1345-1359. |
| [64] | N. a. H. G. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research,* vol. 15, no. 1, pp. 1929-1958, 2014. |
| [65] | A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, “7.5. Batch Normalization,” in *Dive into Deep Learning*, 2020. |
| [66] | Y. Li, *COMP70010 Deep Learning Class Lecture: "Recurrent Neural Networks",* Department of Computing, Imperial College London, 2021. |
| [67] | N. Schuele, “What Are Virtual Instruments,” Pragmatic Sound, Medium, 2019. [Online]. Available: https://medium.com/pragmatic-sound/what-are-virtual-instruments-c762055ddf71. [Accessed June 2021]. |
| [68] | Clavia DMI AB, “Nord Electro 6D,” Nord Keyboards, 2021. [Online]. Available: https://www.nordkeyboards.com/products/nord-electro-6. [Accessed June 2021]. |
| [69] | S. Papetti, F. Avanzini and F. Fontana, “Design and Application of the BiVib Audio-Tactile Piano Sample Library,” *Applied Sciences,* vol. 9, no. 5, 2019. |
| [70] | V. Emiya, N. Bertin, B. David and R. Badeau, “Midi-Aligned Piano Sounds (MAPS) - A piano database for multipitch estimation and automatic transcription of music,” Telecom ParisTech, Département Traitement du Signal et des Images, Paris, France, 2010. |
| [71] | Clavia DMI AB, “Nord Piano Library,” Nord, 2021. [Online]. Available: https://www.nordkeyboards.com/sound-libraries/nord-piano-library/information. [Accessed June 2021]. |
| [72] | S. Garcia-Valencia, A. Betancourt and J. G. Lalinde-Pulido, “Sequence Generation using Deep Recurrent Networks and Embeddings: A study case in music,” *ArXiv e-prints,* vol. abs/2012.0, no. 2012.01231, 2020. |
| [73] | M. Senior, “Phase Demystified: Understanding Phase Cancellation,” Sound On Sound, 2008. [Online]. Available: https://www.soundonsound.com/techniques/phase-demystified. [Accessed June 2021]. |
| [74] | R. Steinmetz and L. Wolf, “Optical Storage Media, CD-DA: Characteristics (Slide 13),” Darmstadt University of Technology, [Online]. Available: https://eclass.uoa.gr/modules/document/file.php/D246/Lectures/cd.pdf. [Accessed June 2021]. |
| [75] | C. Henson, “How To Build A Piano Instrument In KONTAKT,” Spitfire Audio, 2019. [Online]. Available: https://youtu.be/b9mqaBy0Axs?t=326. [Accessed June 2021]. |
| [76] | B. Glocker, *Machine Learning for Imaging Class Lecture: "Fully Convolutional LeNet", in "Image Segmentation", slide 120,* Department of Computing, Imperial College London, 2021. |
| [77] | R. Gong, “Code Repository for 'Timbre Analysis of Music Audio Signals with Convolutional Neural Networks',” GitHub, 2017. [Online]. Available: https://github.com/ronggong/EUSIPCO2017. [Accessed January 2021]. |
| [78] | J. Schlüter, “Learning to Pinpoint Singing Voice from Weakly Labeled Examples,” in *ISMIR*, 2016. |
| [79] | D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” 2014. |
| [80] | Scikit-learn Developers, “Metrics and Scoring: 3.3.2.4. Balanced accuracy score,” Scikit-learn documentation, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/model\_evaluation.html#balanced-accuracy-score. [Accessed June 2021]. |
| [81] | B. Glocker, *Machine Learning for Imaging Class Lecture: "Image Segmentation" (Slide 32),* Department of Computing, Imperial College London, 2021. |
| [82] | Scikit-learn Developers, “3.1. Cross-validation: evaluating estimator performance,” Scikit-learn documentation, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/cross\_validation.html. [Accessed June 2021]. |
| [83] | M. Brookes, “VOICEBOX: Speech Processing Toolbox for MATLAB,” Imperial College London Department of Electrical & Electronic Engineering, [Online]. Available: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html. [Accessed November 2020]. |
| [84] | Librosa development team, “Librosa documentation,” [Online]. Available: https://librosa.org/doc/latest/index.html. [Accessed January 2021]. |
| [85] | NumPy Authors, “Array programming with NumPy,” *Nature,* vol. 585, no. 7825, pp. 357-362, 2020. |
| [86] | A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison and A. Tejani, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024-8035. |
| [87] | Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot and Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011. |
| [88] | N. Plath, “Influence of Playing on the Tonal Characteristics of a Concert Piano - An Observational Study,” in *Proceedings of the International Symposium on Music Acoustics (ISMA) 2019*, Detmold, Germany, 2019. |
| [89] | conTimbre, “conTimbre - Infos,” [Online]. Available: https://www.contimbre.com/en/infos. [Accessed January 2021]. |
| [90] | KeyPleezer, “LivingRoom Upright Piano - Free Edition,” 2021. [Online]. Available: https://keypleezer.com/livingroom-upright-piano/free-edition/. [Accessed June 2021]. |
| [91] | L. Fritts, “Musical Instrument Samples (MIS),” University of Iowa Electronic Music Studios, [Online]. Available: http://theremin.music.uiowa.edu/MIS.html. [Accessed January 2021]. |
| [92] | T. Eerola, R. Ferrer Flores and V. Alluri, “MUMS Revised,” University of Jyväskylä, 2017. [Online]. Available: https://www.jyu.fi/hytk/fi/laitokset/mutku/en/research/projects2/past-projects/coe/materials/mums/MUMS. [Accessed January 2021]. |
| [93] | J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan and M. Norouzi, “Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders,” 2017. |
| [94] | Ivy Audio, “Piano in 162,” 2015. [Online]. Available: https://ivyaudio.com/Piano-in-162. [Accessed June 2021]. |
| [95] | B. Liang, “Dataset for Analysing Effects of Piano Pedalling Techniques,” Zenodo, 2017. [Online]. Available: https://zenodo.org/record/3242149. [Accessed January 2021]. |
| [96] | C. Henson, “Pianobook,” 2020. [Online]. Available: https://www.pianobook.co.uk/. [Accessed January 2021]. |
| [97] | M. Goto, “RWC Music Database: Musical Instrument Sound Collection,” Real World Computing Partnership (RWCP) of Japan, 2003. [Online]. Available: https://staff.aist.go.jp/m.goto/RWC-MDB/rwc-mdb-i.html. [Accessed January 2021]. |
| [98] | G. Sandell, “Sandell Harmonic Archive (SHARC),” Northwestern University, IL, USA, 1991. [Online]. Available: http://gregsandell.com/j/pageSharc.php. [Accessed January 2021]. |
| [99] | G. Beller, “FullSOL (Studio Online Dataset),” IRCAM, 2020. [Online]. Available: https://forum.ircam.fr/projects/detail/fullsol/. [Accessed January 2021]. |
| [100] | E. F. Feichtner and B. Edler, “Description of the Single Note Database SNDB,” in *145th Audio Engineering Society Convention*, New York, NY, USA, 2018. |
| [101] | MathWorks, Inc., “Using MATLAB with Python,” [Online]. Available: https://uk.mathworks.com/products/matlab/matlab-and-python.html. [Accessed January 2021]. |
| [102] | V. Perrault, “Timbre Toolbox,” Github, 2018. [Online]. Available: https://github.com/VincentPerreault0/timbretoolbox. [Accessed January 2021]. |
| [103] | Vienna University of Technology, MIR group, “Rhythm & Timbre Feature Extraction from Music,” 2015. [Online]. Available: http://ifs.tuwien.ac.at/mir/musicbricks/#RPextract. [Accessed January 2021]. |
| [104] | T. Lidy and A. Rauber, “Evaluation of Feature Extractors and Psycho-Acoustic Transformation for Music Genre Classification,” in *6th International Conference on Music Information Retrieval (ISMIR)*, London, UK, 2005. |
| [105] | Music Technology Group, Universitat Pompeu Fabra, “Essentia: Open-source library and tools for audio and music analysis, description and synthesis,” [Online]. Available: https://essentia.upf.edu/. [Accessed January 2021]. |
| [106] | Wikipedia, The Free Encyclopedia, “Compact Disc Digital Audio - Bit rate,” [Online]. Available: https://en.wikipedia.org/wiki/Compact\_Disc\_Digital\_Audio#Bit\_rate. [Accessed June 2021]. |