DEEP NEURAL NETWORKS WITH VOICE ENTRY ESTIMATION HEURISTICS FOR VOICE SEPARATION IN SYMBOLIC MUSIC REPRESENTATIONS

Reinier de Valk
Jukedeck Ltd.
reinier@jukedeck.com

Tillman Weyde
Department of Computer Science
City, University of London
t.e.weyde@city.ac.uk

ABSTRACT

In this study we explore the use of deep feedforward neural networks for voice separation in symbolic music representations. We experiment with different network architectures, varying the number and size of the hidden layers, and with dropout. We integrate two voice entry estimation heuristics that estimate the entry points of the individual voices in the polyphonic fabric into the models. These heuristics serve to reduce error propagation at the beginning of a piece, which, as we have shown in previous work, can seriously hamper model performance.

The models are evaluated on the 48 fugues from Johann Sebastian Bach's *The Well-Tempered Clavier* and his 30 inventions—a dataset that we curated and make publicly available. We find that a model with two hidden layers yields the best results. Using more layers does not lead to a significant performance improvement. Furthermore, we find that our voice entry estimation heuristics are highly effective in the reduction of error propagation, improving performance significantly. Our best-performing model outperforms our previous models, where the difference is significant, and, depending on the evaluation metric, performs close to or better than the reported state of the art.

1. INTRODUCTION

In the domain of symbolic music representation, the term *voice separation* denotes the identification of individual lines (*voices*) in polyphonic music. More formally, it can be defined as "the task of separating a musical work consisting of multi-note sonorities into independent constituent voices" [3]. With regard to the term *voice* itself, whose meaning is left ambiguous in the above definition, a distinction can be made between (i) a voice as a monophonic sequence of successive, non-overlapping notes, and (ii) a voice as a perceptually independent, but not necessarily monophonic, sequence of notes or multi-note simultaneities [3]. The former definition corresponds to the music-theoretical notion of a voice (also *part*) [3, 9], while the latter corresponds to the music-psychological notion of an *auditory stream* [2]. For certain genres of music—e.g., piano sonatas or string quartets—it is more appropriate to think of the polyphonic fabric as consisting of multiple streams that may or may not be (partly) monophonic.

Voice separation, especially in monotimbral polyphonic music (e.g., harpsichord or lute music) for more than three concurrent voices, has been recognised as a difficult task even for professional musicians [15, 16, 30]. From a music information retrieval (MIR) perspective, voice separation is considered a challenge that has not yet been addressed satisfactorily. It is, however, an important task: an adequate identification of the individual voices is a prerequisite for tackling several open MIR and musicological problems, such as automatic transcription [1], pattern retrieval [11, 26, 29], and melodic querying [24, 36].

Over the past decade, deep neural networks (DNNs) have been successfully applied to various computer vision, speech recognition, and natural language processing tasks, and, increasingly, to MIR tasks [5]. Consisting of multiple processing layers, DNNs can learn representations of data with multiple levels of abstraction [25], which makes them better suited than their shallow counterparts to model complex input-output relationships. Despite their successful application to a number of MIR tasks, DNNs have not yet been used for voice separation.

The main contributions of this paper are:

- the implementation and evaluation of DNNs for voice separation in symbolic music representations;
- the implementation and evaluation of improved *voice entry estimation heuristics*;
- the creation of a public benchmark dataset for voice separation, which currently does not exist.

We show that a model that combines a DNN with the voice entry estimation heuristics performs close to or better than the reported state of the art.

In what follows, in Section 2, related work is discussed. In Section 3, the model and the integrating framework are presented, and in Section 4, the evaluation method is explained. Section 5 is dedicated to the voice entry estimation heuristics, and Section 6 to the dataset. In Section 7, the experimental results are discussed, and in Section 8, conclusions and directions for future work are presented.

2. RELATED WORK

The existing models addressing the task of voice separation can be divided into two categories: rule-based models and machine learning models. A characteristic that the models in both categories share is that they almost all lean heavily on at least one of two perceptual principles fundamental in auditory stream segregation, coined the *Pitch Proximity Principle* and the *Principle of Temporal Continuity* in [16]. These principles dictate that the closer two notes are to one another in terms of pitch or time, respectively, the more likely they are perceived as belonging to the same voice.

2.1 Rule-based models

The rule-based models form the largest category, containing a wide array of approaches. In [34], a preference rule system for contrapuntal analysis is presented. *Preference rules* are criteria by which a possible analysis is evaluated. Dynamic programming techniques are used to limit the amount of possible analyses to be evaluated. In later work [35], a probabilistic model of polyphonic music analysis, incorporating a stream segregation component that builds on the earlier work, is introduced. Inspired by [34] is the algorithm presented in [27], which consists of a voice configuration unit generating well-formed local solutions, and a note assignment unit calculating a preferred solution.

In [4], a *contig* mapping approach is presented, in which the music is divided into segments where a constant number of voices is active, the contigs. Starting from the contigs where the number of voices active equals the nominal number of voices, the optimal connections to the neighbouring contigs are determined. Gradually branching out, this process is repeated until all contigs are connected. A modified version of this approach is proposed in [17], where the connection of contigs that share a boundary at which the number of voices increases is prioritised. The idea is that the distinctiveness (in terms of pitch distance) of the new voice will prevent it from being connected incorrectly to one of the voices active in the smaller-size contig. A further improvement of the approach is described in [14], where, taking into account more context information, additional criteria that underly the contig connection policy are proposed. The criteria are weighted using a genetic algorithm with mutation and crossover operators.

In [33], voice separation is modelled as a clustering problem. Using an agglomerative single-link clustering algorithm, in an iterative process that starts from an initial distribution in which each note is a cluster, all clusters are combined into larger clusters until n simultaneous clusters, the voices, remain. In [10], the music is modelled as a directed graph. The goal is to create a set of disjoint paths, the voices, through the graph. To this end, the graph is divided into segments, which are analysed through constraint satisfaction optimisation. Using a sequence alignment algorithm, the analyses are then connected.

Two models stand out as they allow for non-monophonic voices. In the local optimisation approach proposed in [21], a piece is partitioned into slices that are processed iteratively, assigning the notes to voices. A stochastic local search algorithm is used to find assignments that minimise a parametric cost function assessing the assignments; weighting the parameters in a certain way can result in non-monophonic assignments. In the Voice Integration/Segregation Algorithm (VISA) as proposed in [19, 20] and later refined in [31], *vertical integration*—concurrent notes with the same onset and duration merging perceptually into a single sonority—is considered to be prior to *horizontal integration*—successive notes close in pitch and time merging perceptually into a single voice. VISA thus first identifies concurrent notes that merge into single sonorities, and, using a bipartite matching algorithm, then assigns the sonorities to separate streams.

2.2 Machine learning models

In [23], VoiSe, a system for separating voices in both implicit and explicit polyphony, is presented. The system consists of two components: a same-voice predicate implemented as a learned decision tree, which determines whether or not two notes belong to the same voice, and a hard-coded algorithm that maps notes to voices.

A probabilistic, Markov chain-like, system is proposed in [18]. Based on pitch information only, the system learns how likely a note is to occur for a voice, as well as how likely a transition between two notes is to occur. The system is inspired by [4] in that the music is processed in a similar manner—starting at chords in which all voices are present. Another probabilistic approach is described in [6], where the music is represented as a sequence of chords, and a discrete hidden Markov model (HMM) is used to determine the most likely sequence of mappings to voices (the hidden states) for the chords (the observations). A similar, although more sophisticated, approach using an HMM is proposed in [28]. This model explicitly allows notes within a single voice to overlap. This not only makes preprocessing (quantisation) redundant, but also enables application to data generated from live performance.

In [6], the task of voice separation is modelled both as a multi-class classification problem (see also [7]), where the music is represented as a sequence of notes, which are assigned to voices (the classes), and as a regression problem, where the music is represented as a sequence of chords, for which mappings to voices are rated. Standard single-hidden layer feedforward neural networks are used as the classifier and regressor, respectively. In [13], too, the music is represented as a sequence of chords, and a single-hidden layer feedforward neural network is used to greedily assign each chord note to the voice that maximises a trained assignment probability.

3. PROBLEM FORMULATION, MODEL, AND FRAMEWORK

As in [6, 7], in this paper we formulate the task of voice separation as a multi-class classification problem, where each note in a piece is assigned to one of v voices (the classes). We assume that a voice is always monophonic

(see Section 1), and that the number of voices in a piece is equal to its nominal number of voices, which we infer from the size of its largest chord. (These assumptions do not always hold true, but in pure contrapuntal music one generally finds only few exceptions. We resolve such cases by removing the offending notes from the dataset; this is discussed in Section 6.) Furthermore, for practical reasons we set the maximum value of v to 5. This enables us to process pieces containing up to five voices, which currently suffices. The maximum number of voices determines the number of classes and hence the size of the neural network's output layer; for the sake of efficiency, it should thus be kept as small as possible.

3.1 Model

We use the open source TensorFlow machine learning library [1] (version 1.6.0) to implement a multi-layer deep feedforward neural network that uses the rectified linear unit activation function for all $L - 1$ hidden layers and the softmax activation function for the output layer, and that has five output neurons, each representing a class. Given that our dataset is relatively small, we use batch training, where we check the performance on the validation set (comprising every fifth training example) every 10 epochs as early stopping strategy, and store the earliest best-performing model. We use Xavier initialisation [12] for the weights and initialisation with zeros for the biases, the Adam optimisation algorithm [22] to minimise the cross-entropy loss, and dropout [32] to prevent overfitting. We set the learning rate to 0.01 and the number of training epochs to 600, values we observed to work well. Three further hyperparameters are optimised using a grid search (see Section 7): the dropout *keep probability*, the number of hidden layers, and the size of the hidden layers.

3.2 Framework

We integrate the model in our previously developed framework for data preprocessing, feature extraction, and cross-validated training and evaluation [6], implemented in Java. [2] In this framework, the music is represented as a sequence of notes, by default ordered by (i) onset time (low to high) and (ii) pitch (low to high). When evaluating the model, this sequence is processed in linear fashion, where for each note a feature vector is calculated that is given as input to the model, which then makes a class decision—thus assigning the note to a voice.

3.2.1 Feature vector

Each note is represented by a 33-dimensional feature vector, containing properties of that note in its polyphonic context. The features are handcrafted and can be divided into four categories of increasing scope: (i) note-level features, encoding individual properties of the note; (ii) note-chord features, encoding the note's position in the chord; (iii) chord-level features, encoding properties shared by all notes in the chord; and (iv) polyphonic embedding features, encoding the note's polyphonic relation to the notes in the previous as well as the current chord. All feature values (except certain default values) are scaled to fall in the range [0, 1]. An overview is presented in Table 1; more detail is provided in [6].

Index	Feature	Description
0	*pitch*	pitch, as a MIDI number
1	*duration*	duration, in whole notes
2	*isOrnamentation*	true (1) if a 16th note or shorter, false (0) if not
3	*indexInChord*	index (pitch-based) in the chord
4	*pitchDistBelow*	distance to note below
5	*pitchDistAbove*	distance to note above
6	*chordSize*	number of chord notes
7	*metricPosition*	metric position in the bar
8	*numNotesNext*	number of notes (onsets) in the next chord
9-12	*intervals*	intervals in the chord
13-17	*pitchProx*	for each voice v, the pitch proximity to the adjacent left note in v
18-22	*interOnsetProx*	idem, inter-onset
23-27	*offsetOnsetProx*	idem, offset-onset
28-32	*voicesOccupied*	for each voice v, whether it is currently occupied (1) or not (0)

Table 1. The feature vector, containing note-level (0-2), note-chord (3-5), chord-level (6-12), and polyphonic embedding features (13-32). Pitch distances and intervals are measured in semitones; proximities are inverted distances.

4. EVALUATION

We evaluate the models using k-fold cross-validation. Because it is not desirable that identical or highly similar samples extracted from one piece end up in both the training and the test set, we partition a dataset along its individual pieces rather than randomly. k thus equals the number of pieces in a dataset; each piece in it serves as test set once.

4.1 Evaluation metrics

We use four metrics to assess model performance. *Accuracy* is a per-note metric that measures the proportion of notes that have been assigned to the correct voice:

$$\text{acc} = \frac{|C|}{|N|}, \tag{1}$$

where C is the set of notes assigned to the correct voice, and N the set of all notes.

Soundness and *completeness* are complementary metrics that measure transitions between note pairs. We use the definitions provided in [23]. If f is an assigned

[1] https://www.tensorflow.org/

[2] https://www.github.com/reinierdevalk/voice_separation/

voice and g a correct voice, then a pair of adjacent notes (n_t, n_{t+1}) in f is considered *sound* if $g(n_t) = g(n_{t+1})$ holds. (Note that, according to this definition, f and g need not be the same voice.) Extending the definition, we take soundness to be the proportion of sound pairs in *all* voices:

$$\text{snd} = \frac{|S|}{|P|}, \tag{2}$$

where S is the set of sound pairs, and P the set of all pairs in all assigned voices f. Similarly, a pair of adjacent notes (n_t, n_{t+1}) in correct voice g is considered *complete* if assigned voice $f(n_t) = f(n_{t+1})$ holds. We take completeness to be the proportion of complete pairs in all voices:

$$\text{cmp} = \frac{|C|}{|P|}, \tag{3}$$

where C is the set of complete pairs, and P the set of all pairs in all correct voices g.[3]

Average voice consistency (AVC), coined in [4], measures, "on average, the proportion of notes from the same voice that have been assigned ... to the same voice". The *voice consistency* (VC) for voice v is calculated as follows:

$$\text{VC}(v) = \frac{1}{|S(v)|} \max_{u \in V} \{|n \in S(v) : vN(n) = u|\}, \tag{4}$$

where $S(v)$ is the set of notes assigned to v, V the set of all voices, and $vN(n)$ the correct voice for note n. The AVC, then, is the average VC over all voices:

$$\text{AVC} = \frac{1}{|V|} \sum_{v \in V} \text{VC}(v). \tag{5}$$

The per-fold percentages for each metric m are weighted by the number of notes (or note pairs) in the piece for the fold, so that the average values over all folds are always per-note (or per-pair):

$$\text{avg}(m) = \frac{\sum_{i=1}^{k} (m_i \cdot |N_i|)}{\sum_{i=1}^{k} |N_i|}, \tag{6}$$

where k is the number of folds, and N the set of notes in a piece.

4.2 Evaluation modes

We use two evaluation modes: *test mode* and *application mode*. In test mode, the feature vectors are calculated using the correct voice information for the preceding notes. Test mode serves a gauging function in that it reflects the *optimal* model performance on unseen data. In application mode, the feature vectors are calculated using the model-generated voice information. This mode corresponds to the real-world application scenario where no correct voice information is available, and where all voice decisions must be based on previous decisions—it thus reflects the *expected* model performance on unseen data. In application mode, model performance can suffer from *error propagation*.

[3] The definitions are equal to those given for *precision* and *recall* in [10], metrics used in [13, 14, 18, 27, 28]. The terms appear to be used interchangeably.

5. VOICE ENTRY ESTIMATION HEURISTICS

Error propagation is the phenomenon in which an incorrect voice assignment influences the voice decision for the following notes negatively. Given the accuracy in test (acc_T) and application mode (acc_A), the proportion of misassignments due to error propagation, q, is calculated as follows:

$$q = \frac{\text{acc}_\text{T} - \text{acc}_\text{A}}{1 - \text{acc}_\text{A}}. \tag{7}$$

In previous work [6, 7], depending on the dataset we observed q values up to 0.87, indicating that model performance is indeed seriously hampered by error propagation.

Although error propagation can occur throughout a piece, it tends to be particularly strong in thinly-textured openings of pieces, where the model may start 'on the wrong foot'. This often leads to a chain of misassignments. To address this problem, we propose a preprocessing step that applies two heuristics, h1 and h2 (improving on [8]). They estimate which notes belong to the *new voices* at each *density increase*, that is, each point where the maximal number of simultaneous notes so far increases. h1 and, partly, h2 are based on the prior assumptions that (i) voices tend to move in small steps, and that when new voice(s) enter, (ii) none of the already active voices has a rest, and (iii) none of the voices is involved in voice crossing.

```
01  function estimate(list notes) returns list
02    density increases d := [d_1, ..., d_m]
03    available voices av := [1, ..., d_m]
04    add av to new list fw
05    for i from m to 2:
06      if h1: find lowest-cost configuration
07        at pos(d_i)
08      if h2: find pattern at pos(d_i)
09      remove new voices from av
10      prepend av to fw
11      av := copy(av)
12    return fw
```

Figure 1. Algorithm outline. Underlined concepts are explained in the main text.

h1 and h2 share a similar overall algorithmic structure, as shown in Figure 1. The algorithm takes as input the sequence of notes representing a piece (see Section 3.2), and returns, for each density increase (including the opening), a vector of voice assignments for the first chord of the increased density. If the voices enter successively, h2 is called; if not, or if h2 fails, h1 is called. The voice assignments returned remain fixed when the DNN is applied.

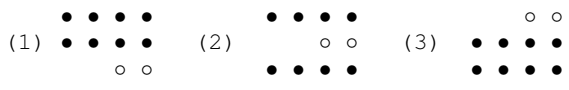

Figure 2. Chord configurations ($n = 2$). Columns represent chords; rows represent layers.

h1 is the more generic heuristic. It determines the new voices by calculating, at each density increase starting at the last, the lowest-cost *configuration*. A configuration organises the last n chords (i.e., ordered sequences of

pitches) of density d_{i-1} and the first n chords of density d_i into horizontal layers, as shown in Figure 2. The cost for a configuration is calculated as follows:

$$\sum_{j=1}^{n}\sum_{k=1}^{d_{i-1}}\sum_{l=1}^{n}|p_{j,k}-p_{l,k}|, \tag{8}$$

where $p_{j,k}$ is the pitch at the kth $2n$-sized layer in the jth chord of density d_{i-1}, and $p_{l,k}$ the pitch at the kth $2n$-sized layer in the lth chord of density d_i. The positions of the remaining n-sized layers in the lowest-cost configuration, then, determine the new voices.

h2 caters specifically to imitative pieces, and attempts to determine the new voices by finding, at each density increase starting at the last, a match for the *pattern* (as defined by the first n notes of the piece, the opening motif's head) in the first n chords of density d_i. The first matching criterion is rhythmic sequence; if this yields multiple matches, melodic contour (up, same, down) is added as a second matching criterion. If a single match is thus found, the new voice is identified; if multiple matches are still found, the lowest-cost configuration (as in h1) is used to disambiguate. If no match is found, which can happen if the prior assumptions do not hold true, the new voice is assumed to enter below the existing voice(s). If at more than half of the density increases no match is found, h2 fails.

6. DATASET

The models are evaluated on the 48 fugues from Johann Sebastian Bach's *The Well-Tempered Clavier* (BWV 846-893), containing one two-, 26 three-, 19 four-, and two five-voice pieces, as well as his 30 inventions (BWV 772-801), containing 15 two- and 15 three-voice pieces (also known as *sinfonias*). The dataset, in MIDI format, was originally retrieved from the MuseData repository of the Center for Computer Assisted Research in the Humanities, [4] and has been slightly modified. First, all *in-voice chords*—instances where a voice is non-monophonic—were reduced to single notes, and all temporarily added extra voices were removed. Figure 3 shows an example of both. Second, because of liberties in performance or rounding errors leading to note overlap within a voice, occasionally some quantisation was required. This was achieved by adjusting each offending left note's offset to equal its adjacent right note's onset. These first two modifications are necessary in order for the data to comply with the assumptions that underly our modelling approach (a voice is always monophonic, and the number of voices in a piece is equal to its nominal number of voices—see Section 3). Third, to create a more equal distribution of training and test data in cross-validation, the two-voice fugue was split into two parts, and the two five-voice fugues were split into four (BWV 849) and two (BWV 867) parts. Fourth, for a number of pieces starting with an anacrusis, some padding with rests was required to ensure a correct metrical alignment. Fifth, where necessary, time signature or key signature information was corrected or added.

[4] http://www.musedata.org/

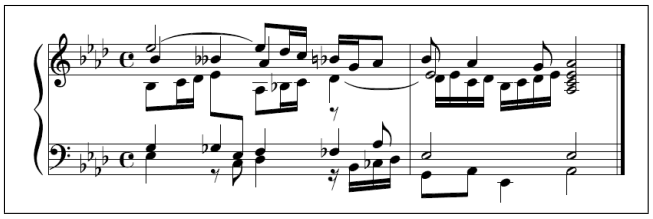

Figure 3. *The Well-Tempered Clavier*, Fugue 17 in A♭ major (BWV 886), closing bars. Temporarily added extra voice, chromatically descending from G_3 to $E\flat_3$ (lower staff), and in-voice chord (upper staff, final chord).

Thus, a total of 206 notes were pruned from the original 53230 notes in the fugues, and a total of five notes from the original 19872 notes in the inventions—yielding a dataset containing 72891 notes. We publish this dataset as a curated benchmark dataset for voice separation, [5] that enables the comparison of results in a rigorous manner, and that thus facilitates reproducible research [37].

7. EXPERIMENTAL RESULTS AND DISCUSSION

In a first experiment, we performed a grid search to optimise three hyperparameters: the number of hidden layers (HL), the size of the hidden layers (HLS), and the value of the dropout keep probability (KP). We explored a small hyperparameter space determined in earlier experimentation, consisting of four HL values (2, 3, 4, and 5), four HLS values (25, 33, 50, and 66), and three KP values (0.75, 0.875, and 0.9375). The grid search was performed on the 19 four-voice fugues; as the deciding metric, accuracy in test mode was used (metrics in test mode are more stable indicators of model performance; see Section 4.2). For each HL value, we selected the best-performing model, which we then trained and evaluated on all 48 fugues and all 30 inventions. This was done separately on the different subsets (two-voice, three-voice, etc.); the performance on all fugues or inventions is the per-note (or per-pair) average over their subsets as calculated using Equation (6). Table 2 shows that on the fugues, the two-layer model yields the highest performance in both test and application mode. On the inventions, the results are less clear—although the two more shallow models seem to perform better here too. Overall, however, the results are fairly similar, indicating a limited effect of the number of layers.

Focussing on the best model (HL = 2; HLS = 66, KP = 0.875), in a second experiment, we then investigated the effect of using a deep(er) neural network, as well as the effect of the integration of the voice entry estimation heuristics. To this end, we compared four models: the single-hidden layer neural network as described in [6, 7] (N), the same model with the heuristics integrated (N/h), the two-layer model (D), and the two-layer model with the heuristics integrated (D/h). The heuristics were not used in test mode, as error propagation does not occur there. Table 3 shows that D always outperforms N, and that N/h and D/h always outperform N and D, respectively. A test for

[5] https://www.github.com/reinierdevalk/data/

HL	HLS	KP	Test				Application			
			acc	snd	cmp	AVC	acc	snd	cmp	AVC
2	66	0.875	98.34	97.11	**97.26**	**98.27**	**90.72**	**96.43**	**96.39**	**90.76**
3	66	0.75	**98.36**	**97.12**	97.24	**98.27**	89.96	96.30	96.30	90.05
4	50	0.75	98.32	97.07	97.23	98.25	89.97	96.36	96.36	90.12
5	50	0.75	98.27	96.98	97.13	98.20	89.96	96.38	96.35	90.23
2	66	0.875	99.09	98.52	98.58	99.06	**96.64**	98.14	98.09	**96.49**
3	66	0.75	99.17	**98.64**	98.62	99.13	96.53	**98.22**	**98.21**	96.35
4	50	0.75	**99.20**	**98.64**	**98.66**	**99.15**	96.54	98.17	98.13	96.34
5	50	0.75	99.00	98.40	98.39	98.96	96.44	97.95	97.93	96.28

Table 2. Experiment 1. Best-performing models per HL value, 48 fugues (top) and 30 inventions (bottom). Values are averages over the different subsets (see Section 7 and Equation (6); all values are percentages.

Model	Test				Application					q
	acc	snd	cmp	AVC	acc	snd	cmp	AVC	F_1	
N	97.86	96.54	96.70	97.78	86.36	95.46	95.33	86.73	95.39	0.84
N/h					90.44	95.86	95.69	90.62	95.78	0.77
D	**98.34**	**97.11**	**97.26**	**98.27**	87.69	96.26	96.20	87.43	96.23	0.86
D/h					**90.72**	**96.43**	**96.39**	**90.76**	**96.41**	0.82
[28]								88.23	97.00	
[17]								89.21		
[10]									92.5	

Table 3. Experiment 2 and 3. N, N/h, D, and D/h models, 48 fugues (top); [10, 17, 28] models, 48 fugues (bottom). Values are averages over the different subsets (see Section 7 and Equation (6); all values except q are percentages. The F_1 score is the harmonic mean of soundness and completeness.

statistical significance (we used the one-tailed Wilcoxon signed-rank test with $p < 0.05$ as the significance criterion) reveals that these performance differences are always significant. We thus conclude that both using a deep(er) neural network and integrating the heuristics yield a significant performance improvement. Furthermore, the q values show that the heuristics indeed reduce error propagation—but the effect is weaker in case of the D model, where error propagation is also slightly worse. Finally, we note that integrating the heuristics leads to a strong improvement in terms of accuracy and AVC. The improvement in terms of soundness and completeness—which are by definition less affected by error propagation—, on the other hand, is only small.

Additionally, we compared the performance of our overall best model (D/h) on the 48 fugues with the performances reported for the three voice separation models that, to our knowledge, represent the current state of the art, and that have also been evaluated on the 48 fugues. As Table 3 shows, D/h outperforms the [17] and [10] models. It also outperforms the [28] model in terms of AVC, but not in terms of F_1 score—which may be because the latter model is specifically optimised for that metric, whereas D/h is optimised for accuracy.

It should be noted, finally, that a strict comparison with the state of the art is problematic due to the heterogeneity of datasets and metrics used. We address this by making our dataset publicly available as a benchmark dataset (see Section 6).

8. CONCLUSIONS AND FUTURE WORK

In this paper, we present the implementation and evaluation of DNNs for voice separation in symbolic music representations as well as the implementation and evaluation of two voice entry estimation heuristics. We evaluate the models on 78 keyboard works by Johann Sebastian Bach, which we publish as a curated benchmark dataset for comparing voice separation models. We observe that both the use of deep(er) neural networks for the task and the integration of the heuristics into the models improve performance significantly. The best model outperforms our previous models, and performs close to or better than the reported state of the art.

A first analysis of the results reveals that the model has difficulties processing musically challenging passages, containing, for example, voice crossings or reduced textures. Furthermore, despite the success of the voice entry estimation heuristics, error propagation remains problematic. An in-depth analysis of the results, planned for future work, is required to gain better insight into these matters. Possible explanations are that the model is not given enough context information, and that it does not have any memory. We therefore also plan to encode a larger polyphonic window into the features as to increase the context information, and we plan to experiment with other types of DNNs, such as recurrent neural networks, which allow information to persist, or long short-term memory models, which are capable of learning long-time dependencies.

9. REFERENCES

[1] E. Benetos, S. Dixon, D. Giannoulis, Automatic music transcription: Challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434, 2013.

[2] A. S. Bregman and J. Campbell. Primary auditory stream segregation and perception of order in rapid sequences of tones. *Journal of Experimental Psychology*, 89(2):244–249, 1971.

[3] E. Cambouropoulos. Voice and stream: Perceptual and computational modeling of voice separation. *Music Perception*, 26(1):75–94, 2008.

[4] E. Chew and X. Wu. Separating voices in polyphonic music: A contig mapping approach. In U. K. Wiil, editor, *Computer Music Modeling and Retrieval: Second international symposium, CMMR 2004*, pages 1–20. Springer, Berlin, 2005.

[5] K. Choi, G. Fazekas, K. Cho, and M. Sandler. A tutorial on deep learning for music information retrieval. *arXiv:1709.04396v1 [cs.CV]*, 2017.

[6] R. de Valk. *Structuring lute tablature and MIDI data: Machine learning models for voice separation in symbolic music representations.* PhD thesis, City University, London, 2015.

[7] R. de Valk and T. Weyde. Bringing 'Musicque into the tableture': Machine-learning models for polyphonic transcription of 16th-century lute tablature. *Early Music*, 43(4):563–576, 2015.

[8] R. de Valk and T. Weyde. Voice entry estimation heuristics to reduce error propagation in voice separation models. In *Proc. of the 18th International Society for Music Information Retrieval Conference, Suzhou, China,* Late-Breaking/Demo session, 2017.

[9] W. Drabkin. Part (ii). In Stanley Sadie, editor, *The new Grove dictionary of music and musicians*, volume 19, page 164. Macmillan, London, 2nd edition, 2001.

[10] B. Duane and B. Pardo. Streaming from MIDI using constraint satisfaction optimization and sequence alignment. In *Proc. of the International Computer Music Conference, Montreal, QC, Canada*, 2009.

[11] M. Giraud, R. Groult, and F. Levé. Subject and counter-subject detection for analysis of the *Well-Tempered Clavier* fugues. In M. Aramaki, M. Barthet, R. Kronland-Martinet, and S. Ystad, editors, *From sounds to music and emotions: 9th international symposium, CMMR 2012*, pages 422–438. Springer, Berlin, 2013.

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. of the 13th International Conference on Artificial Intelligence and Statistics, Sardinia, Italy*, pages 249–256, 2010.

[13] P. Gray and R. Bunescu. A neural greedy model for voice separation in symbolic music. In *Proc. of the 17th International Society for Music Information Retrieval Conference, New York, NY, USA*, pages 782–788, 2016.

[14] N. Guiomard-Kagan, M. Giraud, R. Groult, and F. Levé. Improving voice separation by better connecting contigs. In *Proc. of the 17th International Society for Music Information Retrieval Conference, New York, NY, USA*, pages 164–170, 2016.

[15] D. Huron. Voice denumerability in polyphonic music of homogeneous timbres. *Music Perception*, 6(4):361–382, 1989.

[16] D. Huron. Tone and voice: A derivation of the rules of voice-leading from perceptual principles. *Music Perception*, 19(1):1–64, 2001.

[17] A. Ishigaki, M. Matsubara, and H. Saito. Prioritized contig combining to segregate voices in polyphonic music. In *Proc. of the 8th Sound and Music Computing Conference, Padua, Italy*, 2011.

[18] A. Jordanous. Voice separation in polyphonic music: A data-driven approach. In *Proc. of the International Computer Music Conference, Belfast, Ireland*, 2008.

[19] I. Karydis, A. Nanopoulos, A. Papadopoulos, E. Cambouropoulos, and Y. Manolopoulos. Horizontal and vertical integration/segregation in auditory streaming: A voice separation algorithm for symbolic musical data. In *Proc. of the 4th Sound and Music Computing Conference, Lefkada, Greece*, pages 299–306, 2007.

[20] I. Karydis, A. Nanopoulos, A. N. Papadopoulos, and E. Cambouropoulos. VISA: The voice integration/segregation algorithm. In *Proc. of the 8th International Conference on Music Information Retrieval, Vienna, Austria*, pages 445–448, 2007.

[21] J. Kilian and H. H. Hoos. Voice separation—A local optimization approach. In *Proc. of the 3rd International Conference on Music Information Retrieval, Paris, France*, pages 39–46, 2002.

[22] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980v9 [cs.LG]*, 2017.

[23] P. B. Kirlin and P. E. Utgoff. VoiSe: Learning to segregate voices in explicit and implicit polyphony. In *Proc. of the 6th International Conference on Music Information Retrieval, London, UK*, pages 552–557, 2005.

[24] I. Knopke and F. Jürgensen. A system for identifying common melodic phrases in the masses of Palestrina. *Journal of New Music Research*, 38(2):171–181, 2009.

[25] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[26] D. Lewis, T. Crawford, and D. Müllensiefen. Instrumental idiom in the 16th century: Embellishment patterns in arrangements of vocal music. In *Proc. of the 17th International Society for Music Information Retrieval Conference, New York, NY, USA*, pages 524–530, 2016.

[27] S. T. Madsen and G. Widmer. Separating voices in MIDI. In *Proc. of the 7th International Conference on Music Information Retrieval, Victoria, BC, Canada*, pages 57–60, 2006.

[28] A. McLeod and M. Steedman. HMM-based voice separation of MIDI performance. *Journal of New Music Research*, 45(1):17–26, 2016.

[29] D. Meredith, K. Lemström, and G. A. Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*, 31(4):321–345, 2002.

[30] N. Orio. Music retrieval: A tutorial and review. *Foundations and Trends in Information Retrieval*, 1(1):1–90, 2006.

[31] D. Rafailidis, E. Cambouropoulos, and Y. Manolopoulos. Musical voice integration/segregation: VISA revisited. In *Proc. of the 6th Sound and Music Computing Conference, Porto, Portugal*, pages 42–47, 2009.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(Jun):1929–1958, 2014.

[33] W. M. Szeto and M. H. Wong. Stream segregation algorithm for pattern matching in polyphonic music databases. *Multimedia Tools and Applications*, 30(1):109–127, 2006.

[34] D. Temperley. *The cognition of basic musical structures*. MIT Press, Cambridge, MA, 2001.

[35] D. Temperley. A unified probabilistic model for polyphonic music analysis. *Journal of New Music Research*, 38(1):3–18, 2009.

[36] G. Velarde, T. Weyde, and D. Meredith. An approach to melodic segmentation and classification based on filtering with the haar-wavelet. *Journal of New Music Research*, 42(4):325–345, 2013.

[37] M. D. Wilkinson et al. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(160018), 2016.