AUGMENTEDNET: A ROMAN NUMERAL ANALYSIS NETWORK WITH SYNTHETIC TRAINING EXAMPLES AND ADDITIONAL TONAL TASKS

Néstor Nápoles López
McGill University, CIRMMT
nestor.napoleslopez@mail.mcgill.ca

Mark Gotham
Universität des Saarlandes
mark.gotham@uni-saarland.de

Ichiro Fujinaga
McGill University, CIRMMT
ichiro.fujinaga@mcgill.ca

ABSTRACT

AugmentedNet is a new convolutional recurrent neural network for predicting Roman numeral labels. The network architecture is characterized by a separate convolutional block for bass and chromagram inputs. This layout is further enhanced by using *synthetic training examples* for data augmentation, and a *greater number of tonal tasks* to solve simultaneously via multitask learning. This paper reports the improved performance achieved by combining these ideas. The *additional tonal tasks* strengthen the shared representation learned through multitask learning. The *synthetic examples*, in turn, complement key transposition, which is often the only technique used for data augmentation in similar problems related to tonal music. The name 'AugmentedNet' speaks to the increased number of both training examples and tonal tasks. We report on tests across six relevant and publicly available datasets: ABC, BPS, HaydnSun, TAVERN, When-in-Rome, and WTC. In our tests, our model outperforms recent methods of functional harmony, such as other convolutional neural networks and Transformer-based models. Finally, we show a new method for reconstructing the full Roman numeral label, based on common Roman numeral classes, which leads to better results compared to previous methods.

1. INTRODUCTION

Automatic Chord Recognition (ACR) has been extensively explored in the field of Music Information Retrieval (MIR). ACR systems typically seek to predict the root and quality of the chords throughout a piece of music via either an audio or a symbolic representation. A more specific type of chordal analysis, particularly relevant for Western classical music, is functional harmony. The main difference between ACR and functional harmony is that the latter requires other adjacent tasks to be solved simultaneously, notably including the detection and identification of key changes (modulations [1, 2] and tonicizations [3]).

The analytical process of functional harmony is often described through Roman numeral annotations. This annotation system is particularly popular in Western music theory for the analysis of 'common-practice' tonal music. Roman numeral annotations encode a great deal of information about tonality, in a compact syntax. For instance, an annotation like **C:vii^{o6}/V** encodes the local key (C-major), quality of the chord (diminished triad), chord inversion (first), and any existing tonicization (G-major).

This 'modular' nature of Roman numeral annotations has been beneficial to MIR research. In recent years, functional harmony has been approached by dividing the main task in several sub-tasks. Thus, as a machine learning problem, functional harmony can be expressed as the task of correctly predicting sufficient sub-tasks to reconstruct the full Roman numeral label. Furthermore, recent work in the standardization and conversion of Roman numeral analyses has provided MIR researchers with a larger metacorpus of annotations for training new models [4, 5]. Yet, despite these developments and the growing interest in the field, the performance of functional harmony models for predicting Roman numeral labels remains relatively low.

In this paper, we propose a new neural network architecture that improves the prediction of functional harmony and its relevant features. Besides the architecture itself, our model benefits from increased data augmentation (beyond key transpositions), and an additional set of output tasks that enhance the effects of multitask learning demonstrated by other researchers [6]. To facilitate the work of other researchers, we release all of our preprocessed datasets, data splits, experiment logs, and the full source code of our network in `https://github.com/napulen/AugmentedNet`.

2. RELATED WORK

For a summary of general ACR strategies, see Pauwels et al. [7]. Here we focus on prior work in the specific area of automatic functional harmonic analysis.

The first computational works on Roman numeral analysis were by Winograd [8] and Maxwell [9]. Later, the independent contributions by Temperley, Sleator, and Sapp led to the first end-to-end automatic Roman numeral analysis system: a program named *Melisma* [10–12]. Notable subsequent studies include Raphael and Stoddard [13], Illescas et al. [14], and Magalhães and de Haas [15], who proposed Hidden Markov Models (HMMs), dynamic programming, and grammar-based approaches, respectively.

More recently, deep neural networks have become the

preferred tool for approaching this problem. Chen and Su [6] were the first to introduce 'multitask learning' (MTL) [16] to the problem as a suitable way for the neural network to share representations between related tonal tasks. Chen and Su's model consists of a bidirectional LSTM [17] followed by task-specific dense layers, which implement the MTL configuration. In this work, the authors also introduced the 'Beethoven Piano Sonata Functional Harmony' dataset for evaluating such models. The MTL layout outperformed single-task configurations, and it has continued to be the best-performing approach in subsequent deep learning studies. Recently, the same authors have adopted Transformer-based networks to deal with functional harmony and ACR [18,19]. The work with these networks has explored the capability of attention mechanisms to improve the performance of ACR, paying special consideration to chord segmentation and its evaluation.

Micchi et al. [20], in turn, proposed a convolutional recurrent neural network (CRNN). The recurrent component consists of a bidirectional GRU [21] connected to task-specific dense layers, similar to those of Chen and Su [6]. In their experiments, a DenseNet-like [22] convolutional component outperformed other configurations (e.g., dilated convolutions or a GRU with pooling). Micchi et al. also demonstrated the positive effect of using pitch spelling in the inputs and outputs. This confers at least two advantages: it provides a more informative output (e.g., not only the correct key, but the correct spelling between two enharmonic keys), and it increases the theoretical number of transpositions available for data augmentation.

Here, we propose improvements along the line of CRNNs. Due to our focus on extended data augmentation and tonal tasks, we named our network *AugmentedNet*.

3. AUGMENTEDNET

The AugmentedNet is a similar network in size and design to the one by Micchi et al. [20]. It is characterized by a different layout of the convolutional layers, a new representation of pitch spelling, and a separation of the bass and chroma inputs into independent convolutional blocks.

3.1 Inputs

Reference note per timestep. The input to the network consists of a sequence of timesteps, which are sampled from the score at symbolically regular note duration values. In this study, we use the thirty-second note ('demisemiquaver') as this atomic value (i.e., eight timesteps per quarter note in the score), in order to match the most fine-grained frame sampling seen in previous work. The length of the sequence is set by a fixed number of timesteps. Following Micchi et al., we set that number at 640 frames (or 80 quarter notes) per sequence example.

Bass and spelled chroma features. The representation of each timestep is conceptually the same as in Micchi et al. [20], a vector containing a spelled bass note and spelled chroma features. However, the length of our vectors is different. In the Micchi et al. representation, each timestep has 70 features: 35 for the bass and 35 for the chroma features. We consolidate this information in 38 features: 19 for the bass, and 19 for the chroma features. The reduction in number of features is due to an alternative encoding of pitch spelling, which we explain below.

Encoding the pitch spelling. We split the representation of a pitch spelling into two components: the pitch class (0–12) and the generic note letter (A–G). Each spelled pitch thus leads to a two-hot encoded vector with 19 features (1 of 12 pitch classes, and 1 of 7 note names). This reduces the number of parameters in the network without any observable compromise in performance. Furthermore, the spelled bass and chroma inputs are connected to the network separately, in their own convolutional blocks. The input to each block is a tensor of pitch spelling sequences.

3.2 Convolutional block

Using the feature maps of previous layers as an input to a convolutional layer has proven beneficial, for instance, by strengthening feature propagation and reducing the number of parameters [22]. Moreover, DenseNet-like architectures have shown to work well for the specific task of functional harmony [20].

We follow similar methods, reusing the feature maps computed for a given timestep in subsequent convolutions. Figure 1 provides a schematic diagram of our network, with the convolutional block on the top left area of the figure. In our preliminary experiments, we noticed that different tonal tasks have different time dependencies. For example, losing information about a specific timestep often leads to poor performance in predicting the inversion, whereas losing long-term context hinders the performance in key estimation. Our architecture implicitly prioritizes short-term dependencies in the initial convolutional layers, by having more filters and covering less timesteps. Going further, the convolutions provide more context about future timesteps, but output less filters. These increments (in window size) and decrements (in number of filters) are done in powers of 2. Using six convolutional layers in each block (as shown in Figure 1), the first layer convolves a window of a single timestep (a thirty-second note), whereas the sixth layer utilizes a window of 32 timesteps (a whole note). The output shape of each block is the original length of the sequence, with 82 features per timestep.

3.3 Dense and recurrent layers

Two time-distributed dense layers are applied to the concatenated outputs of the convolutional blocks. The dense layers help to reduce the number of features before the GRU layers. These have 64 and 32 neurons, respectively.

Two bidirectional GRU [21] layers are applied after the second dense layer. Both GRU layers return outputs at every timestep. Throughout the entire network, the dimensionality of the timesteps axis remains constant. That is, our input and output sequences have the same length, and the model predicts one Roman numeral label per timestep.

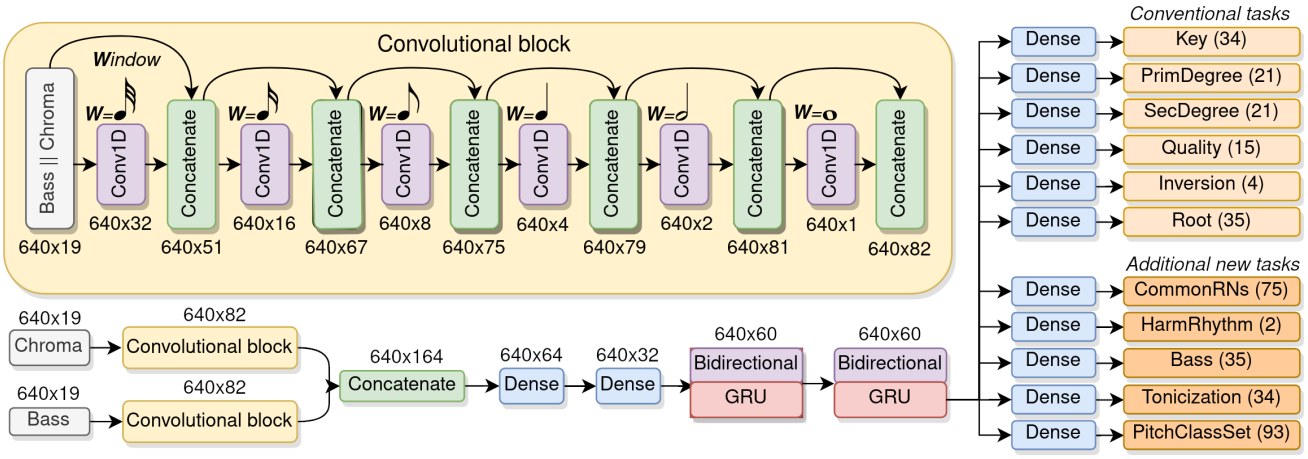

Figure 1. *AugmentedNet*. The bass and chroma inputs are processed through independent convolutional blocks and then concatenated. Both convolutional blocks are identical and expanded on the top of the figure. A convolutional block has six 1D convolutional layers. Each layer doubles the length of the convolution window and halves the number of output filters. On the right, the MTL layout with eleven tasks. Each task indicates the number of output classes in parentheses.

3.4 Multitask learning

The output of the network follows an MTL approach with hard parameter sharing, similar to the one by Chen and Su [6]. For each of the output tasks, a time-distributed dense layer is attached to the second GRU and used to predict its corresponding task. In the past, MIR researchers have reconstructed Roman numeral annotations using six tasks: the (local) key, the primary and secondary degrees, the chord quality, the chord inversion, and the chord root [6, 20]. In our network, these six 'conventional' tasks are learned as well, plus five new additional ones. All eleven tasks and their number of output classes are shown on the right side of Figure 1.

All the conventional tasks, except for the key, have the same number of output classes described by Micchi et al. [20]. The key includes four additional classes: {F♭, G♯, d♭, e♯}. These were included because our dataset, larger than previous ones, revealed modulations reaching G♯ major. Thus, the number of allowed key signatures was extended by one sharp and one flat, in both modes.

3.4.1 Five additional new tasks

It is argued that MTL may improve the performance of a model by preferring representations that are useful to related tasks, acting as an implicit form of data augmentation and regularization method [16]. Roman numeral labels can be divided into different parts, of which the six conventional tasks are known examples. Motivated by the prospective improvement of our network, we included five new additional tasks, which have relevance to harmonic analysis. One of these tasks, CommonRNs, was used to design an alternative method to reconstruct the full Roman numeral label. The remaining four are included to strengthen the shared MTL layers. We hypothesize that these additional tasks (e.g. pitch class sets) improve the accuracy of the model because of the MTL layout, even if they are not explicitly used to predict the Roman numeral.

CommonRNs: during data exploration, we found that, when inversions were removed and synonyms (e.g., $\flat\mathbf{II}_6$ and $\mathbf{N}_6$) were standardized, a set of 75 Roman numeral classes spanned ~98% of all the annotations across the training set (see Table 1). This was a motivation to predict these classes directly as an additional task. The correct prediction of this new task is equivalent to predicting the primary and secondary degrees, chord root, and chord quality simultaneously. As an additional experiment (see Section 4.3), we tested an alternative new method to reconstruct the Roman numeral labels, using the key, inversion, and CommonRNs tasks. We refer to this method as RN_{alt}.

1–15	16–30	31–45	46–60	61–75
I	**V/V**	**Ger**	**vii^{o7}/v**	**V+**
V^7	**v**	**N**	**vii^{o7}/iii**	**vii^o/vi**
V	**V^7/ii**	**vii^{o7}/vi**	**IV/V**	**III+**
i	**III**	**V/ii**	**I+**	**V/iii**
IV	**ii^{ø7}**	**vii^{ø7}**	**I^7**	**ii/V**
ii	**iii**	**V^9**	**vii^o/IV**	**I/♭VI**
vi	**ii^o**	**vii^o/ii**	**V/III**	**vii^{o7}/IV**
iv	**vii^o/V**	**V/iv**	**V^7/iii**	**V^7/v**
vii^{o7}	**V^7/vi**	**Cad/V**	**vii^o/iv**	**i^7**
vii^o	**VII**	**iv^7**	**ii^{o7}**	**iii^7**
V^7/V	**vii^{o7}/ii**	**vii^{o7}/iv**	**VI^7**	**Fr**
V^7/IV	**I/V**	**IV^7**	**I/III**	**V/IV**
vii^{o7}/V	**V^7/iv**	**V^7/III**	**V^7/VI**	**vii**
VI	**V/vi**	**vii^{ø7}/V**	**♭VII**	**V/v**
ii^7	**vi^7**	**It**	**♭VI**	**II**

Table 1. *CommonRNs*. The 75 most-common Roman numeral classes found across the training set. Note that the inversion has been omitted and learned as a separate task.

Harmonic Rhythm: a binary classification task that indicates whether a Roman numeral annotation starts at a given timestep. It may be relevant for chord segmentation.

Bass: a multiclass classification task that indicates the bass note in the Roman numeral label. This task has 35 output classes representing a pitch spelling, as in the chord

root task [20]. It is an alternative chord inversion task.

Tonicization: a multiclass classification task that indicates a tonicized key implied by the Roman numeral label (if any). The output classes are 34 keys, as in the key task, and it is an alternative way to learn the secondary degrees.

Pitch Class Sets: a multiclass classification task that indicates the set of pitch classes implied by the Roman numeral chord. The number of classes (93) results from computing all pitch class sets in all diatonic triad and seventh chords, plus all augmented sixth chords in all keys. This task is related to the chord quality, primary degree, and to non-chord tones [23].

3.5 Data augmentation

3.5.1 Transposition

As in most automatic tonal music analysis research, we transpose each piece to different keys as a form of data augmentation. Particularly, we transpose to all the keys that lie within a range of key signatures, in both modes. When we transpose a piece, we verify that all the modulations within the piece fall in the target range of key signatures. This process of transposition and data augmentation was introduced and described by Micchi et al. [20]. In our data exploration, we found G♯ major to be the furthest key to the center of the *line-of-fifths* [24] in the training set. Thus, we transposed each piece across the keys with 8-flats and 8-sharps in their key signatures.

3.5.2 Synthetic data

In addition to transposition, we implemented a variation of a previous data-augmentation technique by Nápoles López and Fujinaga [25]. Starting with the Roman numeral analyses of our dataset, we synthesized 'new' training examples by realizing the chords implied by each Roman numeral annotation. The synthesis was done using the music21 Python library [26], which converts RomanText [4] files into scores of *block chord* realizations.

We found the default block chord texture of the synthetic examples to be only slightly beneficial for the model, possibly because it did not capture the complex texture of real keyboard music, for example. In order to account for this difference, we artificially 'texturized' the generated training examples, departing from the default block chords. The texturization was done by applying three note patterns recursively (see Figure 2). These patterns were designed intuitively, pursuing certain goals in the resulting texture.

Bass-split (measure 1): a pattern where the original chord duration is divided by half, playing the bass in the first half, and the remaining notes in the second. The goal is to occasionally separate the bass from all other notes.

Alberti bass (measure 2): a 4-note melodic pattern with a pitch contour of *low-high-middle-high*. The goal is to occasionally play chords using a monophonic texture.

Syncopation (measure 3): a pattern where the highest note is played first, followed by the rest of the notes, played in syncopation. The goal is to occasionally shift the onset of the bass from the onset of the Roman numeral label.

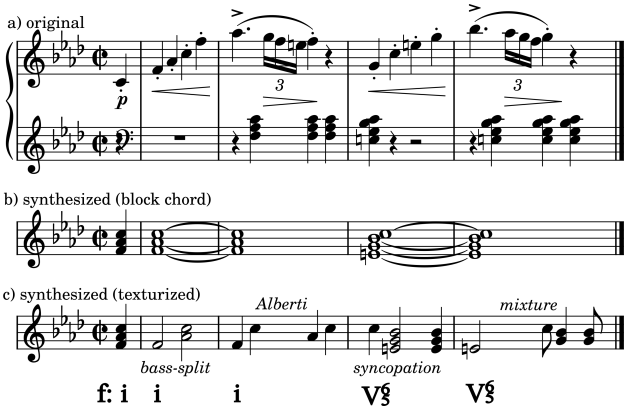

Figure 2. An example of texturization. The *block chord* texture (b) was synthesized using music21 [26] from an input RomanText file [4]. The texturized output (c) was generated by recursively applying note patterns to the block chord scores. The three musical patterns of *bass-split*, *Alberti bass*, and *syncopation* are indicated in measures 1–3, respectively. The original music score (a) is shown for reference: mm. 1–4 of Beethoven's Piano Sonata Op.2 No.1.

Mixture (measure 4): we applied the three patterns randomly and recursively. For example, the *mixture* in measure 4 displays a *bass-split* pattern over the whole-note chord, followed by a *syncopation* pattern applied over the three upper notes, in the second half of the measure.

As part of the randomization, some chords were left unaltered (e.g., the anacrusis of Figure 2), and the patterns were applied across different duration values. To constrain the depth of the recursion, we applied these patterns only to the slices of the score that contained 3–4 simultaneous notes. This process resulted in the generation of 'new pieces' that showed improvements in the learning process of the model, further than the block chord synthetic scores.

4. EXPERIMENTS

4.1 Datasets

We ran experiments using six datasets: Annotated Beethoven Corpus (ABC) [27], Beethoven Piano Sonatas (BPS) [6], Haydn "Sun" Quartets (HaydnSun) [28], Theme and Variation Encodings with Roman Numerals (TAVERN) [29], When-in-Rome [1] (WiR) [4, 5], and the Well-Tempered Clavier (WTC) [4]. Table 2 shows a summary of all datasets. The summary indicates the number of files in each split and the number of sequences (each of 640 frames) that were collected from that split.

For all datasets, the same procedure was followed regarding data splits. Training, validation, and test splits were produced randomly (except in BPS, where they were provided by Chen and Su [6]). Preliminary experiments were conducted in the training set, using the validation set

[1] Note that, in practice, WiR is also a meta-collection and standardization effort, where several of these datasets (e.g., TAVERN) have been converted into a common representation. Here, we list the academic sources of the datasets. For the annotation files, please refer to the relevant literature [4, 5] as well as the accompanying source code of this paper.

Dataset	Files (Seqs) Training	Validation	Test
ABC [27]	50 (448)	10 (97)	10 (99)
BPS [6]	18 (155)	7 (75)	7 (82)
HaydnSun [28]	16 (91)	4 (19)	4 (19)
TAVERN [29]	38 (404)	8 (68)	8 (64)
WiR [4, 5]	107 (301)	21 (61)	21 (51)
WTC [4]	12 (25)	6 (13)	6 (14)
Total	241 (1424)	56 (333)	56 (329)

Table 2. The functional harmony datasets used in our experiments. The splits were generated randomly (except for BPS). For each split, the number of files and the number of sequences (in parentheses) are indicated.

to assess the performance, adjust the hyperparameters, and inform the design of the network architecture. The best-performing version of our model was run once in the test set, this time including the validation portion as part of the training. The results obtained for all the rows labeled *Full dataset* in Table 4 report the results obtained on the corresponding test split.

Data augmentation. For every training example, we synthesized and texturized an additional file, using only the Roman numeral annotations (and ignoring the original score). The original and texturized training examples were transposed to different keys for further data augmentation. Both forms of data augmentation were applied to the training set of a particular experiment, leaving the validation and test sets intact, in order to prevent any data leakage.

4.2 Training procedure

Epochs. We set a fixed number of 100 epochs in all experiments. We found that the use of early stopping was unreliable to determine the end of the training process. Instead, we saved the weights after each epoch. At the end, we selected the weights that maximized the mean accuracy across the six conventional tasks.

Other hyper-parameters. Each of the layers in the network is accompanied by batch normalization [30] before the activation function. In the recurrent layers, we apply the batch normalization after the activation function. All convolutional and dense layers use the rectified linear unit (ReLU) as their activation function. However, the two GRU layers use a hyperbolic tangent. In all of our experiments, we used 16 sequences per batch and the *rmsprop* optimizer [31], with a learning rate of 10^{-3}.

Computing time. The network was trained on a personal laptop [2] with a Linux operating system, Tensorflow v2.4.1 [32], and GPU acceleration. With these hardware and software conditions, the training times are approximately 30 minutes (BPS only), 40 minutes (BPS+WTC), and 250 minutes (Full dataset). The number of trainable parameters in the network is close to 90,000. This number already includes all the parameters introduced by the additional output tasks. Therefore, the model is similar in size to recent approaches [19, 20].

[2] Intel i7 10750h, Nvidia RTX 2070, 32 GB DDR4.

Model	Key	Deg.	Qual.	Inv.	Root	RN
AugN_{6}	82.7	64.4	76.6	77.4	82.5	43.3
AugN_{6+}	83.0	65.1	77.5	**78.6**	83.0	44.6
AugN_{11}	81.3	64.2	77.2	76.1	82.9	43.1
AugN_{11+}	**83.7**	**66.0**	**77.6**	77.2	**83.2**	**45.0**

Table 3. Average accuracy (in %) of four configurations of our model, where $\{6, 11\}$ indicate the number of MTL tasks and '+' indicates the use of synthetic training data.

4.3 Results

AugmentedNet configurations. Table 3 summarizes the performance of different AugmentedNet configurations. For example, with or without the additional tasks, and with or without synthetic data. The configurations were trained on each of the six datasets individually, for a total of 24 experiments. The accuracy reported is the average accuracy obtained by each model configuration across all six datasets. Based on the reported accuracy values, the $\text{AugmentedNet}_{11+}$ (with additional tasks and synthetic training examples) is the best-performing configuration. Thus, in subsequent experiments, we compare this configuration against the current state-of-the-art models.

In the past, functional harmony models have been evaluated using the Beethoven Piano Sonatas (BPS) dataset [6, 18, 20]. The most recent model [19] has also been evaluated using the Well-Tempered Clavier (WTC) dataset [4]. We provide direct comparisons in these two datasets, in so far as that is possible, replicating the experimental conditions of the previous models. In addition, we also report the results of our model across the remaining datasets.

Beethoven Piano Sonatas. The last rows of Table 4 show the results on the BPS dataset. Single lines in the table delimit experiments that are directly comparable. For example, the upper rows of BPS compare the results of Micchi et al. [20] using all of their available training data and our model using the larger dataset available to us now.

Well-Tempered Clavier. Above the BPS rows, in Table 4, we show the evaluation on the WTC dataset. The CS21 model presented the results over 4-fold cross validation [19]. We replicate this study for direct comparison. [3] In these rows, we report the average accuracy across the four folds, as well as the standard deviation.

The results show that our model outperforms both the recent convolutional methods [20] and Transformer-based ones [19] in the reconstruction of the full Roman numeral labels. For ABC, HaydnSun, TAVERN, and WiR, we show the generalization of our model when using all the training data available on the corresponding test set. Finally, we show the overall performance of our model in a composite test set that includes all six datasets (first row of Table 4).

[3] But we used our test split for the *Full dataset* experiment in WTC.

Test set	Training set	Model	Key	Degree	Quality	Invers.	Root	ComRN	RN_{conv}	RN_{alt}
Full test set	Full dataset	AugN	82.9	67.0	79.7	78.8	83.0	65.6	46.4	51.5
WiR	Full dataset	AugN	81.8	69.2	85.9	90.3	90.3	70.2	56.4	62.4
HaydnSun	Full dataset	AugN	81.2	62.9	80.2	82.7	86.5	60.4	48.6	52.1
ABC	Full dataset	AugN	83.6	65.6	78.0	76.9	78.9	62.6	44.5	48.4
TAVERN	Full dataset	AugN	88.7	60.0	77.4	78.8	81.5	66.3	42.6	52.9
WTC	Full dataset	AugN	77.2	69.7	75.0	74.4	82.7	61.7	**46.2**	47.9
$WTC_{crossval}$	BPS+WTC	AugN	**85.1**$_{(4.0)}$	62.9$_{(5.5)}$	69.1$_{(1.9)}$	70.1$_{(3.7)}$	79.2$_{(1.8)}$	59.9$_{(3.4)}$	**42.9**$_{(4.2)}$	46.9$_{(4.7)}$
$WTC_{crossval}$	BPS+WTC	CS21	56.3$_{(2.5)}$	-	-	-	-	-	26.0$_{(1.7)}$	-
BPS	Full dataset	AugN	**85.0**	**73.4**	**79.0**	**73.4**	84.4	68.3	**45.4**	49.3
BPS	All data	Mi20	82.9	68.3	76.6	72.0	-	-	42.8	-
BPS	BPS+WTC	AugN	**82.9**	70.9	80.7	72.0	85.3	67.6	**44.1**	47.5
BPS	BPS+WTC	CS21	79.0	-	-	-	-	-	41.7	-
BPS	BPS	AugN	**83.0**	**71.2**	**80.3**	**71.1**	84.1	68.5	**44.0**	47.4
BPS	BPS	Mi20	80.6	66.5	76.3	68.1	-	-	39.1	-
BPS	BPS	CS19	78.4	65.1	74.6	62.1	-	-	-	-
BPS	BPS	CS18	66.7	51.8	60.6	59.1	-	-	25.7	-

Table 4. Accuracy of five functional harmony models: Chen and Su (2018, 2019, and 2021), Micchi et al. (2020), and $AugmentedNet_{11+}$. In the WTC test set, the comparison against CS21 replicated the 4-fold cross validation [19]. In this case, the standard deviation is indicated in parentheses. For all other rows, the results report the performance on the held test set. The values in the RN_{alt} column indicate the performance using an alternative method for reconstructing the full Roman numeral, as explained in Section 3.4.1.

The RN_{conv} and RN_{alt} methods. As discussed in Section 3.4.1, it is possible to use the 75 most-common Roman numeral classes as an alternative task to the chord root, chord quality, and primary and secondary degree tasks. Thus, an alternative resolution of the Roman numeral label (RN_{alt}) is presented in the last column of the AugmentedNet results. This accuracy corresponds to the reconstruction of the full Roman numeral using the key, chord inversion, and CommonRNs. We found that this, in fact, leads to better results compared to the conventional method (RN_{conv}), which reconstructs the full Roman numeral labels using the six conventional tasks. Table 4 shows the accuracy values for both methods. For completeness, we also show the accuracy of the CommonRNs output task. For this task, note that the maximum achievable accuracy is ~98%, because any class that is not present in the set of 75 CommonRNs will be misclassified.

In summary, the *Full dataset* rows show the best results achieved by our model for each dataset. In all cases, the results of our model are higher than existing methods in the final reconstruction of the Roman numeral label. Additionally, the best results in the reconstruction are achieved via the RN_{alt} method, instead of the conventional one (RN_{conv}).

5. CONCLUSION

We present advances in the use of CRNNs to predict Roman numeral labels. In Beethoven Piano Sonatas (BPS) and the Well-Tempered Clavier (WTC) datasets, our network outperforms existing models in the prediction of the conventional tonal tasks and the reconstruction of the full Roman numeral labels. Furthermore, we demonstrate that the use of an additional task, CommonRNs, is helpful to achieve better results in the final reconstruction step, compared to the conventional method used in previous research. Although we present these general improvements in accuracy, we have not yet assessed the chord segmentation of our model, leaving that for future work. Furthermore, we consider that several ideas presented here may be useful in future automatic tonal music analysis research, notably: (1) the use of additional tonal tasks in functional harmony and, (2) the use of texturized synthetic training examples. Although five additional tasks were presented, there are more potential tasks that can be examined, such as triad vs. seventh classification, tonal function (T, D, and SD), or cadence detection. Our method for synthesizing 'new' training examples applied three note patterns to *texturize* block chords. We developed this method based on observation and music theory domain-knowledge. A more sophisticated approach could offer better texturization outputs. We consider this to have the most potential impact on functional harmony research, because the data is still scarce and expensive to annotate. Although current models have yet to reach the expectations of MIR researchers and musicologists alike, we hope that this goal is not too far. Through the use of new techniques, deep learning models may soon achieve unprecedented results in complex music analytical processes, such as Roman numeral analysis.

6. ACKNOWLEDGMENTS

This research has been supported by the Social Sciences and Humanities Research Council of Canada (SSHRC) and the Fonds de recherche du Québec–Société et culture (FRQSC). We would also like to thank user *ClassicMan* from the MuseScore community, who allowed us to use their MusicXML scores of all Piano Sonatas by Beethoven. This saved us a great deal of time during this research.

7. REFERENCES

[1] L. Feisthauer, L. Bigo, M. Giraud, and F. Levé, "Estimating keys and modulations in musical pieces," in *Proceedings of the Sound and Music Computing Conference*, 2020.

[2] H. Schreiber, C. Weiss, and M. Müller, "Local key estimation in classical music recordings: A cross-version study on Schubert's Winterreise," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020, pp. 501–505.

[3] N. Nápoles López, L. Feisthauer, F. Levé, and I. Fujinaga, "On local keys, modulations, and tonicizations: A dataset and methodology for evaluating changes of key," in *Proceedings of the 7th International Conference on Digital Libraries for Musicology*, 2020, pp. 18–26.

[4] D. Tymoczko, M. Gotham, M. S. Cuthbert, and C. Ariza, "The RomanText format: A flexible and standard method for representing roman numerial analyses," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2019, pp. 123–129.

[5] M. Gotham and P. Jonas, "The openscore lieder corpus," in *Poster at the Music Encoding Conference*, 2021.

[6] T.-P. Chen and L. Su, "Functional harmony recognition of symbolic music data with multi-task recurrent neural networks," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2018, pp. 90–97.

[7] J. Pauwels, K. O'Hanlon, E. Gómez, and M. Sandler, "20 Years of automatic chord recognition from audio," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2019.

[8] T. Winograd, "Linguistics and the computer analysis of tonal harmony," *Journal of Music Theory*, vol. 12, no. 1, pp. 2–49, 1968.

[9] H. J. Maxwell, "An expert system for harmonizing analysis of tonal music," in *Understanding Music with AI: Perspectives on Music Cognition*. Cambridge, MA, USA: MIT Press, 1992, pp. 334–353.

[10] D. Temperley, *The Cognition of Basic Musical Structures*. MIT Press, 2004.

[11] D. Sleator, "The melisma music analyzer," https://www.link.cs.cmu.edu/music-analysis/, Accessed: 2021-07-02.

[12] C. Sapp, "tsroot manpage," http://extras.humdrum.org/man/tsroot/, Accessed: 2021-08-01.

[13] C. Raphael and J. Stoddard, "Functional harmonic analysis using probabilistic models," *Computer Music Journal*, vol. 28, no. 3, pp. 45–52, 2004.

[14] P. R. Illescas, D. Rizo, and J. M. I. Quereda, "Harmonic, melodic, and functional automatic analysis," in *Proceedings of the Sound and Music Computing Conference*, 2007.

[15] J. P. Magalhães and W. B. de Haas, "Functional modelling of musical harmony: an experience report," *ACM SIGPLAN Notices*, vol. 46, no. 9, pp. 156–162, 2011.

[16] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv:1706.05098*, 2017.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[18] T.-P. Chen and L. Su, "Harmony Transformer: Incorporating chord segmentation into harmony recognition," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2019, pp. 259–267.

[19] ——, "Attend to chords: Improving harmonic analysis of symbolic music using Transformer-based models," *Transactions of the International Society for Music Information Retrieval*, vol. 4, no. 1, 2021.

[20] G. Micchi, M. Gotham, and M. Giraud, "Not all roads lead to Rome: Pitch representation and model architecture for automatic harmonic analysis," *Transactions of the International Society for Music Information Retrieval*, vol. 3, pp. 42–54, 2020.

[21] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv:1406.1078*, 2014.

[22] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[23] Y. Ju, N. Condit-Schultz, C. Arthur, and I. Fujinaga, "Non-chord tone identification using deep neural networks," in *Proceedings of the 4th International Workshop on Digital Libraries for Musicology*, 2017, pp. 13–16.

[24] D. Temperley, "The line of fifths," *Music Analysis*, vol. 19, no. 3, pp. 289–319, 2000.

[25] N. Nápoles López and I. Fujinaga, "Harmonic reductions as a strategy for creative data augmentation," in *Late-Breaking Demo at 21st International Society for Music Information Retrieval Conference*, 2020.

[26] M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2010, pp. 637–642.

[27] M. Neuwirth, D. Harasim, F. C. Moss, and M. Rohrmeier, "The Annotated Beethoven Corpus (ABC): A dataset of harmonic analyses of all Beethoven string quartets," *Frontiers in Digital Humanities*, vol. 5, 2018.

[28] N. Nápoles López, "Automatic harmonic analysis of classical string quartets from symbolic score," Master's thesis, Universitat Pompeu Fabra, 2017.

[29] J. Devaney, C. Arthur, N. Condit-Schultz, and K. Nisula, "Theme and variation encodings with Roman numerals (TAVERN): A new data set for symbolic music analysis." in *Proceedings of the International Society for Music Information Retrieval Conference*, 2015, pp. 728–734.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning*, 2015, pp. 448–456.

[31] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv:1609.04747*, 2016.

[32] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016, pp. 265–283.