Article

Automatic Neume Transcription of Medieval Music Manuscripts using CNN/LSTM-Networks and the segmentation-free CTC-Algorithm

Christoph Wick * and Frank Puppe

Chair for Artificial Intelligence and Applied Computer Science, University of Würzburg, 97074 Würzburg, Germany; frank.puppe@uni-wuerzburg.de (F.P.)

* Correspondence: christoph.wick@informatik.uni-wuerzburg.de

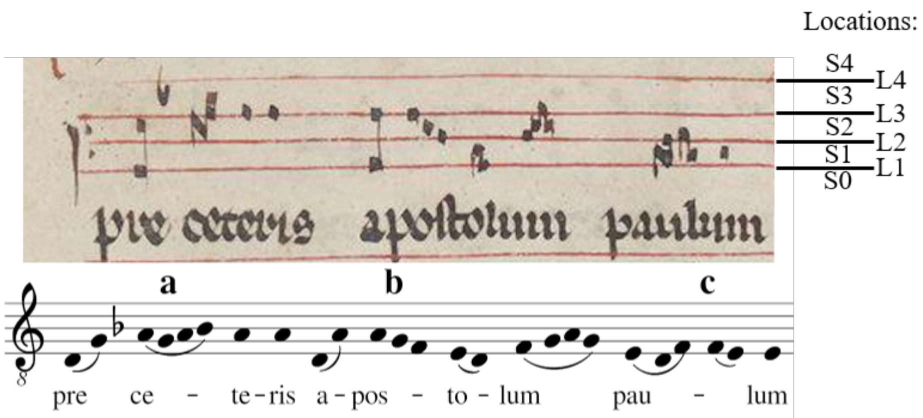
Abstract: The automatic recognition of scanned Medieval manuscripts still represents a challenge due to degradation, non standard layouts, or notations. This paper focuses on the Medieval square notation developed around the 11th century which is composed of staff lines, clefs, accidentals, and neumes which are basically connected single notes. We present a novel approach to tackle the automatic transcription by applying CNN/LSTM networks that are trained using the segmentation-free CTC-loss-function which considerably facilitates the Ground Truth (GT)-production. For evaluation, we use three different manuscripts and achieve a diplomatic Symbol Accuracy Rate (dSAR) of 86.0% on the most difficult book and 92.2% on the cleanest one. To further improve the results, we apply a neume dictionary during decoding which yields a relative improvement of about 5%.

Keywords: Optical Music Recognition; Historical Document Analysis; Medieval manuscripts; neume notation; CNN, LSTM, CTC

1. Introduction

Musicology still relies heavily on manual processes to transcribe manuscripts especially if historic notations, such as the Medieval square notation, are researched. To reduce the time expenditure required to obtain the transcripts, an application of novel automatic approaches using state-of-the-art methodology of artificial intelligence is promising. An applicable automatic transcription would allow to produce great amounts of annotated data which can then be used for large-scale data analysis, such as musical grammars [see e.g., 1] or the comparison of melodies [see e.g., 2]. Medieval neume notations are special because they are similar to modern monophonic Common Western Music Notation (CWMN) in the sense that they already use staff lines and clefs to define the pitch of notes. Notes are always part of larger components called "neumes" which historically originate from a stroke which depicts a short movement of music. In contrast to even earlier notations, the square notation uses discrete square-shaped Note Components (NCs). An example line is shown in Figure 1 which also includes a modern transcription equivalent (second line): Each NC corresponds to a note, notes within a neume that are graphically connected are drawn with a slur, NCs that belong to the same note, but are not connected are rendered with a smaller space. The lyrics of the line, whose syllables can help to differentiate between two neumes or a single neume with two gapped NCs, is also shown.

In this paper, we propose a methodology that is well-established in the field of Automatic Text Recognition (ATR): We tackle the transcription of a music staff as sequence-to-sequence problem where the image of a staff is the input and the sequence of music symbols represents the output (bottom of Figure 1). The method of choice is a combination of a Convolutional Neural Network (CNN) and a bidirectional LSTM [3] architecture which is trained by the so-called Connectionist Temporal Classification (CTC)-algorithm [4]. This setup is well-known from the application in ATR [see e.g., 5,6]. In contrast to Fully Convolutional Networks (FCNs) that are trained with a traditional cross-entropy-loss, the advantage is that the GT is allowed to be segmentation-free, that is, the



clef.F-L2, note.start-L1, note.loop-S2, accid.flat-S3, note.start-L3, note.loop-S2, note.loop-L3, note.loop-S3, note.start-L3, note.start-L3, note.start-L1, note.loop-L3, note.start-L3, note.gap-S2, note.gap-L2, note.start-S1, note.loop-L1, note.start-L2, note.loop-S2, note.loop-L3, note.loop-S2, note.start-S1, note.loop-L1, note.loop-L2, note.start-L2, note.loop-S1, note.start-S1

Figure 1. An original line (first image) and its transcription equivalent (second line) rendered in a modern notation. Looped NCs are visualised by a slur (a), the start of a new neume is indicated by a wider space (b), gapped NCs are notated with a small space (c). The last line shows the label sequence of the input line which is the Ground Truth (GT) for the proposed method. A single token consists of the symbol type (note, clef, or accidental), its subtype (c- or f-clef for clefs; looped, gapped, or note start for notes; flat or sharp for the accidentals), and the position on the staff lines (see position labelling). Based on the clef and the positions of the notes it is possible to obtain the melody of the transcription equivalent.

sequence of music symbols is sufficient while the accurate pixel-positions are not required. This leads to a simplification of the cumbersome process of GT-production because the actual pixels positions of the notes are irrelevant for further processing steps. It is even possible to include existing transcriptions that only accounted for the music symbols to obtain a modern transcription equivalent.

In this paper, we claim the following contributions:

- We are the first to apply a segmentation free CNN/LSTM-hybrid network trained using the CTC-algorithm to transcribe Medieval square notation.
- We compare the proposed approach to our previous one based on FCNs [see 7] and obtained slightly improved results on the same dataset.
- We improve the prediction by about 5% using three different CTC-decoders which incorporate a dictionary of neumes.
- We evaluate the proposed method on three datasets with more than 21,000 symbols in total and achieve a dSAR of 86.0% on the most difficult book and 92.2% on the cleanest one.
- We provide a thorough error analysis.

The remainder of this paper is structured as follows: First, we present publications related to Optical Music Recognition (OMR) on historical manuscripts and those that apply similar methods on CWMN. Afterwards, the used datasets are introduced before our proposed method is described. Finally, we present and discuss our results and conclude the paper by giving an overview of future work.

2. Related Work

A combination of CNNs with LSTMs in an encoder/decoder approach known from the sequence-to-sequence task of machine translation was proposed by van der Wel and Ullrich [8]:

The CNN/LSTM-based encoder processes the input line to obtain a vector as digital representation of the score. Then, the decoder which is only based on LSTMs predicts a sequence of pitches, durations, and finally a stop mark. The evaluation dataset consisted of perfectly rendered scores from the MuseScore Sheet Music Archive[1] and provided 108 pitch and 48 duration categories. Using data augmentation, a pitch and duration accuracy of 81% and 94% was achieved, respectively. Therefore, the total accuracy of notes was 80%.

Calvo-Zaragoza *et al.* [9] proposed a CNN/LSTM-hybrid architecture combined with the CTC-loss function to tackle the automatic recognition of printed monophonic scores in CWMN which is equivalent to our proposed approach for Medieval manuscripts. For evaluation, they published their so-called PrIMuS (Printed Images of Music Staves) dataset which comprises 87,678 real-music incipts rendered by Verovio [10], a web-based music engraver. Their approach required a distinct label for each combination of pitch and rhythm, which is why in total up to 1,781 different classes were required. The evaluation yielded a diplomatic Symbol Error Rate (dSER) of approximately 1%. In contrast, even though our fundamental approach is equivalent, we propose a considerably different network architecture specialised to square notations which also includes an adapted data representation. In general, our task is clearly more difficult because compared to the ideally rendered CWMN music, we work with historical manuscripts written in a completely different notation.

To process single lines of handwritten or rendered music, Baró *et al.* [11] also combined CNNs and LSTMs to predict the pitch and rhythm of notes and other musical symbols (e.g., rests or clefs). Their approach was able to not only recognise monophonic, but homophonic music [see e.g., 12], that is, multiple notes can occur at the same time with the same duration. Thereto, they predicted two probability maps, one for symbols and one for pitches with 80 and 28 classes, respectively. In contrast to a CTC-based approach, training required segmented symbols, not just the plain sequence. Data in such form was available in the rendered PrIMuS and handwritten MUSCIMA++ [13] datasets, both written in CWMN. They achieved a total dSER (both rhythm and pitch must be correct) of 0.3% and 54.5% when evaluating on PrIMuS and MUSCIMA++, respectively. The differences indicate the huge gap between the recognition accuracy of handwritten and engraved music even in the case of mono- or homophonic music.

In the context of an automatic transcription of historical material of monophonic music only a few recent attempts have been published. Ramirez and Ohya [14] proposed a pipeline to automatically transcribe square notation on manuscripts of the 14th to 16th century. Their dataset comprised 136 pages of the Digital Scriptorium repository[2] and contained 847 staves and over 5,000 neumes. First, a brute-force algorithm detected staves by matching a staff template of four staff lines with varying positions, scale, and orientation. In total, 802 staves could be correctly detected (recall of 95%). Next, a pattern matching algorithm detected possible symbol locations with an accuracy of 88%. Finally, a Support Vector Machine was applied to classify the symbols into different neume types whereby an accuracy no lower than 92% was achieved. A further splitting of the neumes into NCs was not performed.

Calvo-Zaragoza *et al.* [15] applied a Hidden Markov Model with a constructed n-gram language model to transcribe line images of mensural notation of the 17th century. They built up a corpus of 576 staves and 13,863 individual symbols comprising notes, rests, or clefs. There were 16 different symbols shapes which could occur on the different discrete vertical locations relative to the staff lines. Combining the shape and the positional information resulted in about 200 different classes. Their best model used a 3-gram language model and achieved an dSER of 40.4%.

[1] https://musescore.com
[2] https://digital-scriptorium.org

Table 1. Overview of the dataset statistics. The bigger Nevers dataset is split manually into three parts that share similarities in layout or handwriting. Only five pages are available in the Assisi dataset whereas 15 double pages are available in Cambrai.

Dataset	Pages	Staves	S.-Lines	Symbols	Notes	Clefs	Accid.
Nevers 1	14	125	500	3,911	3,733	153	25
Nevers 2	27	313	1,252	10,794	10,394	361	39
Nevers 3	8	72	288	1,666	1,581	84	1
Nevers Total	49	510	2,040	16,371	15,708	598	65
Assisi	5	50	200	1,333	1,276	53	4
Cambrai	29	176	704	3,500	3,304	185	11

3. Datasets

To evaluate our proposed method, we use three different manuscripts written in square notation. An overview of the dataset statistics is given in Table 1, example images are shown in Figure 2. The largest dataset which was already introduced in [7] consists of 49 pages of the manuscript "Graduel de Nevers" (accessible at the Bibliothèque nationale de France[3]). The pages were manually split into tree parts that share a similar layout: Part 1 has the best notation quality that hardly suffers from bleeding through, has straight staff lines, and its neumes are clear and distinct. The notation of the second part is very dense, suffers from bleeding, which is why it is the most difficult notation in the Nevers dataset. The third part contains some unclear neumes and very wavy staff lines.

The Assisi dataset is comparatively small and comprises only five annotated pages of the full manuscript. The scans are available at the Italian Digital Library[4] and have large white margin which we did not crop in our experiments. In general, the notation is equal to the second part of the Nevers dataset, but with a considerably higher quality. The Cambrai dataset available at the Catalogue of illuminated manuscripts[5] comprises 15 double pages (29 single pages containing music in total) of the 15th and 16th century. The scans are only available in greyscale due to a reproduction using a microfilm and show a very poor quality. The notation is clean, but pages clearly suffer from degradation, artefacts, bleeding through, and distortions which is why this material is the most complicated one compared to our other datasets.

Based on all pages, we created a dictionary of available neumes. While in total there are 20,288 NCs which are part of 12,852 neumes, only about 518 unique neumes with respect to the location of the first NC are present. If the initial pitch is ignored and therefore only the relative motion of the neume is taken into account, only about 231 neumes are present whereby the longest neume comprises nine NCs.

4. Methodology

In this section, we first describe a general workflow in which the proposed symbol detection algorithm can be embedded. Then, we introduce the algorithm including the network architecture and decoders in detail.

3 https://gallica.bnf.fr/ark:/12148/btv1b8432301z
4 http://www.internetculturale.it
5 http://initiale.irht.cnrs.fr/codex/9353

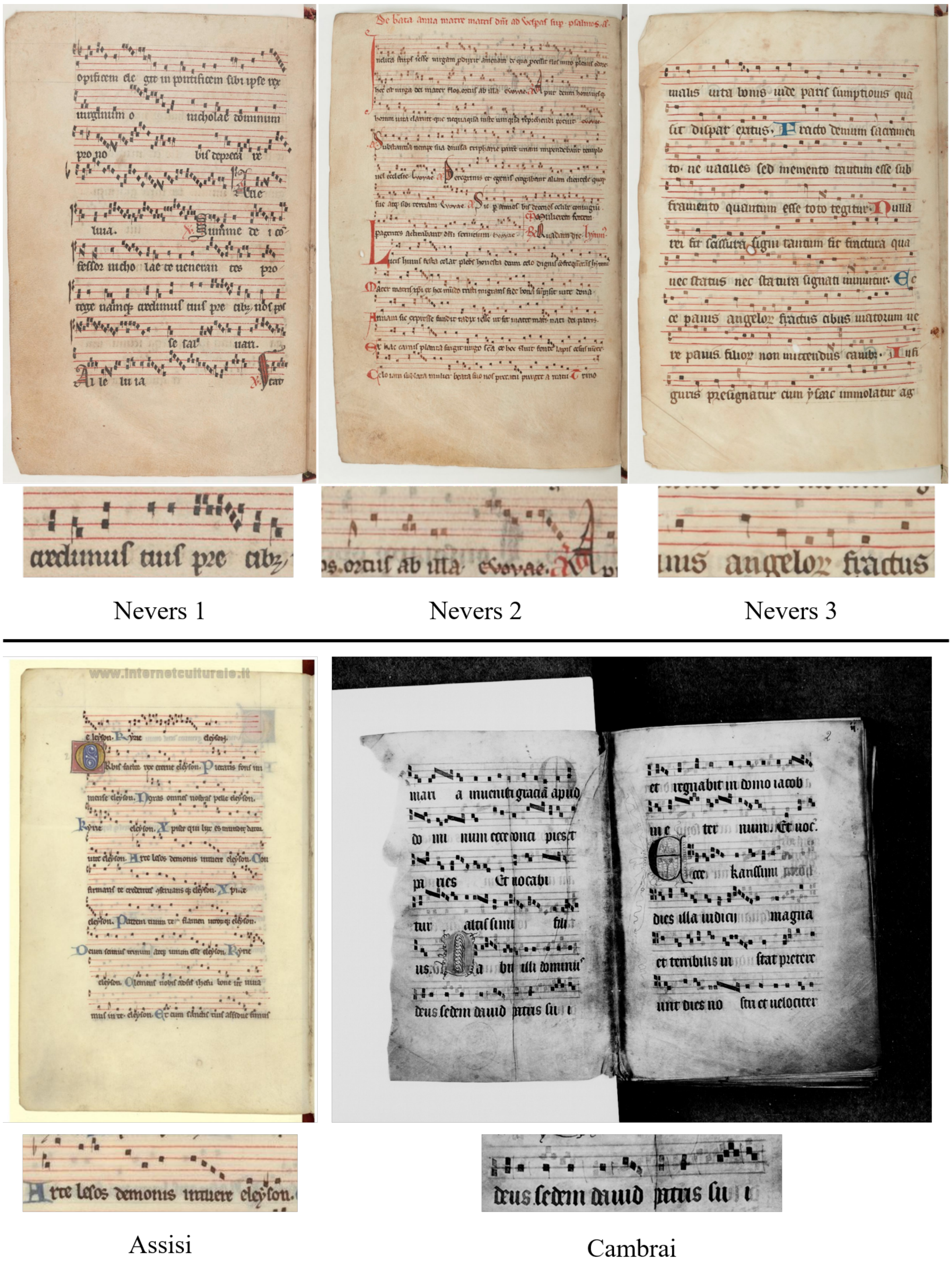


Figure 2. Example pages and a zoomed line of the used datasets. One page for each of the three parts of the Nevers dataset, one of Assisi, and a double page of Cambrai is shown. Note that the white periphery of the Assisi page was manually cropped.

4.1. OMR Workflow

The proposed symbol detection algorithm can be easily integrated into an OMR workflow for transcribing Medieval notations [see 16]. First, the raw scans are preprocessed which produces a deskewed and normalised version of the image. During the normalisation, the average staff line distance d_{SL} is either automatically computed or manually defined to obtain images with a known scale. The images are then processed by a staff line and stave detection algorithm based on FCNs. Based on the bounds of the single staff lines, a complete staff can be extracted to obtain the input for the symbol detection. The next steps are the application ATR on the extracted lyric lines and to finally assign the transcribed syllables to each neume. This paper solely focuses on the symbol detection and thus uses the given staff lines of the datasets.

4.2. Symbol Detection Algorithm

The input of the network is an image of a single staff which we extract based on the already encoded staff lines. First, the whole page is dewarped by computing an image in which all staff lines are straight (see Figure 3). Then, the bounding boxes of all staves are determined and extended by adding a margin of one d_{SL}. Since clefs are often written in the left exterior of the actual lines, we extend the bounds to the left by an additional $3d_{\mathrm{SL}}$. Finally, the line image is extracted and centred in a separate image. Dewarping and centring are important steps for our algorithm because the vertical pixel position then roughly corresponds to the vertical position of a note. An example of an extracted line is shown at the bottom of Figure 3.

An image is then processed sequentially by a CNN/LSTM-hybrid network in horizontal direction. The output is a probability map which concatenates a distribution of all possible labels at each point in time. The labels are defined by the alphabet which are the letters in the case of ATR. In our case, the construction of valid labels is extended because both the type of a music symbol and its vertical position must be encoded. Therefore, we first define six basic types of music symbols (see Figure 4): C-clef, F-clef, Flat-accidental, and the three different types of NCs being either a neume start, gapped, or looped. Each of the two clefs can be located on one of the four staff lines, which is why in total $2 \cdot 4 = 8$ labels are required to define a clef. Accidentals (theoretically) and NCs can occur at any discrete location in a staff, sometimes even on the first lower or upper ledger line. Therefore, in total there are eleven possible vertical positions, and $(1 + 3) \cdot 11 + 8 = 52$ possible labels in total. For an example labelling, see the bottom line of Figure 1.

The implementation of the neural network, and its training and decoding is provided by the open-source ATR-engine Calamari [see 17] which we extended for our task. Only the preprocessing to obtain the deskewed image and thus straight staves, and the evaluation had to be separately written.

4.2.1. CNN/LSTM Network

Figure 5 shows the chosen network architecture based on several preliminary experiments (see Section 5.3.1). First a convolutional layer with 40 kernels with a size of 3×3 is applied to the input image, an image of a staff, to extract the most basic features. The following pooling layer aims to reduce the resolution by only selecting the most prominent features in a 1×2 area. Another two sets of convolution (60 and 80 kernels) and max-pooling layers are appended. The last pooling layer uses a filter size of 2×2, though. Traditional architectures usually solely perform 2×2 pooling halving both axes, we, instead, apply two pooling-layers that only halve the height by the pooling, not the width. The reason is that the maximum number of labels that can be predicted is at most the dimension of the temporal axis of the last layer, which is the width in our case. Since the NCs are sometimes written very dense and also on top of each other, we only inserted one 2×2 pooling-layer. After the last pooling layer, a forth convolutional layer is added with a kernel size of 80, and a bidirectional LSTM-layer with 100 hidden nodes. Finally, a dense layer with a softmax activation converts the 100 output nodes at each position into the 53 (52 + blank) target probabilities at each horizontal position.

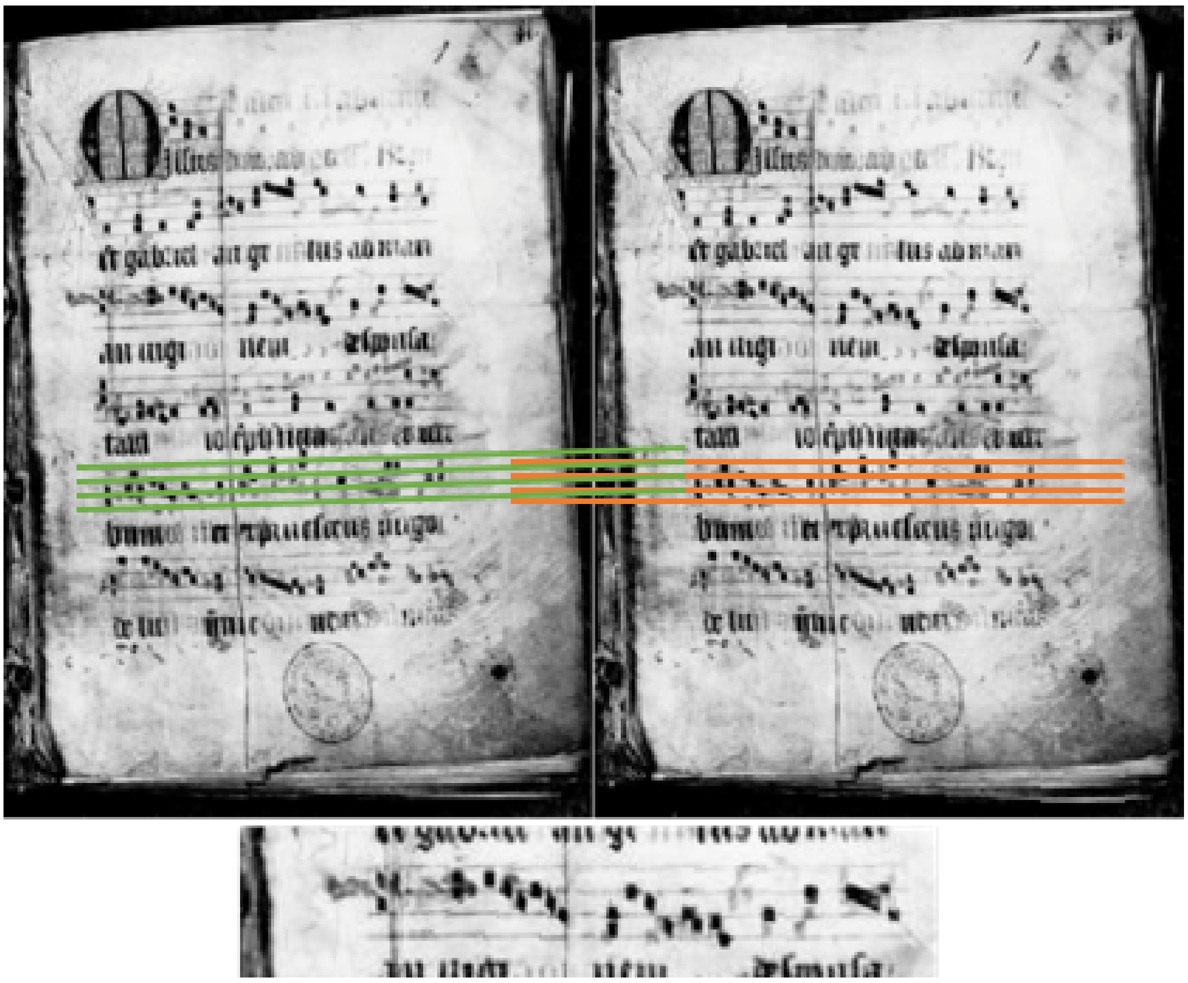


Figure 3. The image shows the pipeline of the line extraction: First the complete page (left) is dewarped based on the staff lines (right), then each single line is cut out (bottom). The overlay of green and orange staff lines emphasise the process. Note that only the right page of a double page of the Cambrai data set is shown.

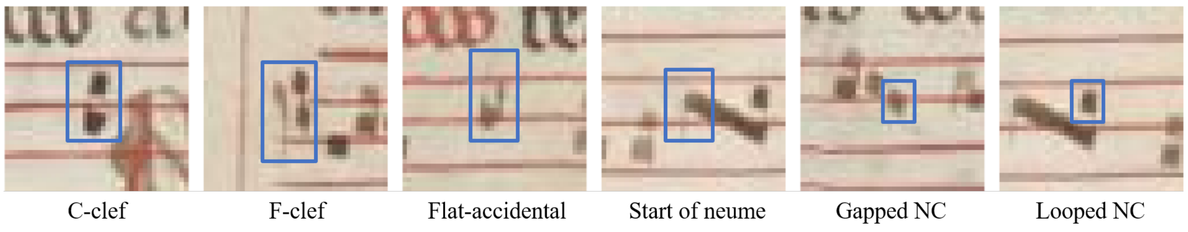


Figure 4. Examples of the six classes to be recognised by the symbol detection.

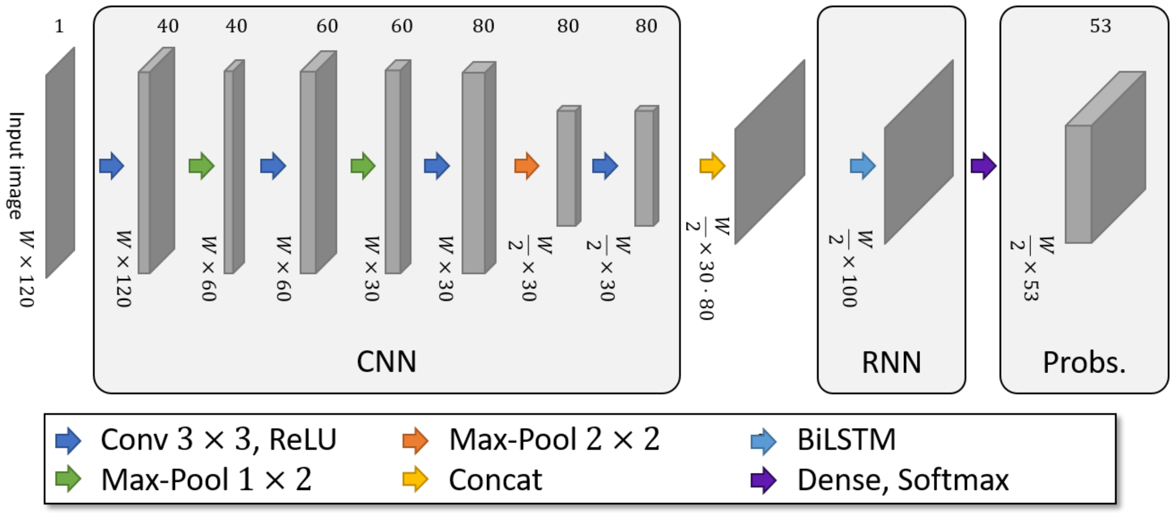


Figure 5. The proposed network architecture. The numbers above each intermediate output denotes the number of channels and therefore the number of kernels in the previous convolutional layer.

4.2.2. CTC-Algorithm and Decoders

The CTC-algorithm allows the network to make "no" prediction at a horizontal position which is enabled by extending the actual alphabet by a so-called *blank* label (in this paper denoted by a "$-$"). The network is trained by providing a loss-function that aims to maximise the probability of the target GT-sequence. The advantage of this algorithm is that the GT-sequence is not segmented, that is, the actual position of a letter, here a music symbol, is not required to be annotated. Instead, the network learns the concept of a symbol and automatically learns to internally segment the input itself. Therefore, the production of GT has a high speed-up because in the case of ATR only the plain text must be typed. In our case of OMR, GT production is still more tedious because the keyboard is not designed to input neume notation. However using keybindings and a sophisticated editor, for example similar to Monodi+ [18], the transcription process can be simplified.

To decode the output of the neural network, we use several algorithms which are directly provided by Calamari: The greedy decoder is the simplest one that first takes the label with the maximum value at each point in time. Then, repeated labels are joined and finally all blanks are erased. For example, if the most probable labels are $--AA--AB$ each character denotes a distinct NC, the resulting decoded sequence is AAB. This decoder has the disadvantage, that any possible sequence is allowed as output. However, as seen in the datasets, the number of different neumes is highly restricted which is why it is sensible to force the decoder to only produce "valid" neumes. This is similar to speech decoding, where only words of a predefined dictionary are permitted.

Therefor, besides the greedy decoder, we apply two different approaches for finding the most probable label sequence given a dictionary. The CTC-Token-Passing (CTC-TP)-algorithm is adopted from Graves [19] and uses the implementation of Scheidl *et al.* [20][6] which we integrated into Calamari. The fundamental idea of the CTC-TP-algorithm is to decode the Recurrent Neural Net (RNN) output combined with dictionary words (in our context neumes) by searching the most likely sequence of dictionary words. Thereto, the characters of each word are modelled as a state machine with intermediate blanks. At each time step (horizontal axis), so-called *tokens* successively store the probabilities for each state of a word. Equally, transitions to new words are allowed by adding the initial state of word and choosing the current most probable word sequence as history. The time complexity of this algorithm is $\mathcal{O}(T \cdot W^2)$, where W is the dictionary size and T the sequence length.

[6] https://github.com/githubharald/CTCDecoder

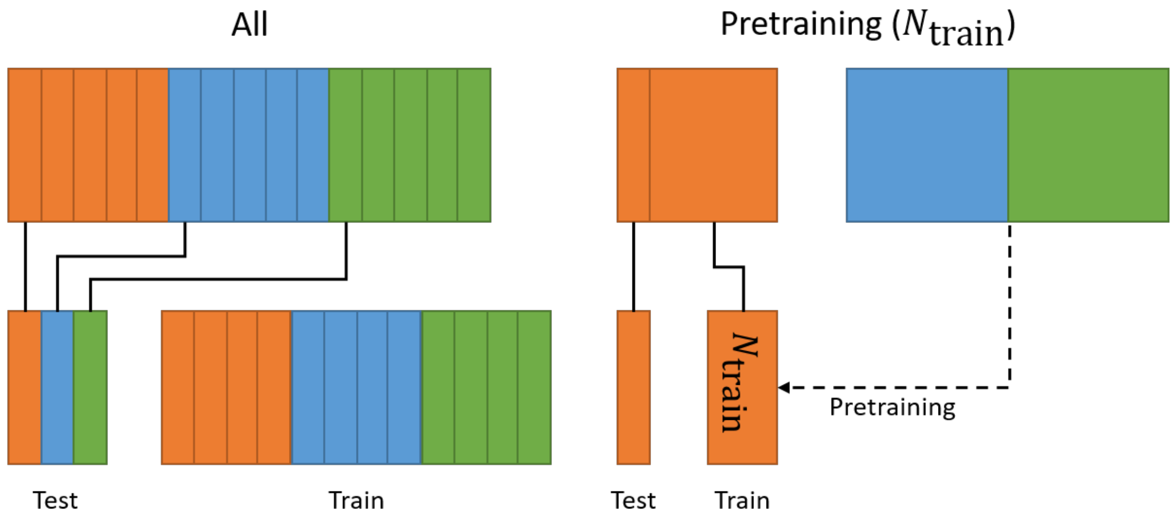


Figure 6. The two schemes used for the training on the different parts of the Nevers dataset. Each colour represents one part which is again split into five folds (smaller boxes). In the experiments, either a slice of all parts are chosen for evaluation and the remainder for training, one part is taken for evaluation and training and both other parts for creating a pretrained model.

The second algorithm is the CTC-Beam-Search (CTC-BS)-algorithm [based on 21] which is most commonly applied for CTC-decoding if a language model shall be integrated. In Calamari, we implement the algorithm according to [20][7] which was extended to allow for word transitions without a space. The beam search stores a list of paths called *beams* that are extended at each time step by all possible further labels (including the blank) according to the language model. This forms a very deep tree which is pruned by only keeping the n_{bm}, called *beam width*, most probable beams. The time complexity if this algorithm is $\mathcal{O}\left(n_{\text{bm}} \cdot C \cdot \log\left(n_{\text{bm}} \cdot C\right)\right)$, where C are the number of characters by which the beams are extended.

5. Experiments

In this section, we first introduce the data setup defining the splitting of training and testing data. Then, we evaluate and discuss the proposed symbol detection algorithm by using the different datasets, comparing it with an FCN-based approach, and incorporating a dictionary.

5.1. Data Setup

Figure 6 shows the setup when training on the three different parts of the Nevers dataset. In any experiment, we apply a cross-fold scheme which splits each part randomly into five smaller slices. In the so-called "All" case, we choose one slice of each part to build the evaluation set, any other part is chosen for training. In the second scenario, we pretrain a model on two complete parts and take one slice of the remaining part for evaluation. The remaining four parts serve as pool of training material, whereby the number of training examples N_{train} is varied to evaluate its impact on the accuracy. If setting $N_{\text{train}} = 0$, we evaluate the performance of the pure pretrained model based on different but somewhat similar material. The overall setup is used to evaluate the proposed approach with our previous one presented in [7].

The two other books (Assisi and Cambrai) are taken to evaluate how well a pretrained model (on the Nevers dataset) performs on other books with a considerably different layout, but which are

[7] https://github.com/githubharald/CTCWordBeamSearch

Table 2. Preliminary experiments for the symbol detection using different network architectures: The number of kernels with a fixed size of 3×3 of convolutional layers (C), the kernel and filter size for max-pooling layers (Mp), and the number of hidden notes for a LSTM layer are stated. The results of the harmonic Symbol Accuracy Rate (hSAR), diplomatic Symbol Accuracy Rate (dSAR), and Neume Accuracy Rate (NAR) given in percent are computed using all tree parts of the Nevers dataset.

ID	Network architecture	hSAR	dSAR	NAR
NA1	C(40), Mp(2x2), C(60), Mp(2x2), C(80), LSTM(100)	89.8	86.2	79.9
NA2	C(40), Mp(2x2), C(60), Mp(1x2), C(80), LSTM(100)	90.6	86.8	80.4
NA3	C(40), Mp(1x2), C(60), Mp(1x2), C(80), LSTM(100)	90.9	87.1	80.4
NA4	C(40), Mp(1x2), C(60), Mp(1x2), C(80), Mp(2x2), C(80), LSTM(100)	91.7	88.3	82.3
NA5	C(40), Mp(1x2), C(60), Mp(1x2), C(80), Mp(1x2), C(80), LSTM(200)	91.5	87.9	81.6

also written in square notation. Thereto, we also vary N_{train} to measure the impact of the number of training instances. This setup equates to the pretraining scenario in Figure 6.

5.2. Metrics

To evaluate the performance of the proposed algorithm, we use three sequence based metrics. The dSER computes the edit-distance of the predicted label sequence and the GT and normalises the result by the maximum length of the two sequences. Missing, additional, and replaced (e.g., with a wrong type) symbols therefore count as one error. The dSAR is the corresponding accuracy rate computed by $1 - \text{dSER}$. The harmonic Symbol Accuracy Rate (hSAR) only evaluates the correctness of the harmonic properties, that is the melody, by ignoring the graphical connections of all NCs. Finally, we introduce the Neume Accuracy Rate (NAR) which only takes complete neumes into account and ignores all clefs and accidentals. The metric is rather strict, because if a single NC is wrong for example due to position or graphical connection, the complete neume is counted as one error. In context of ATR this metric can be compared to a Word Accuracy Rate (WAR) which is then used to evaluate the language model, here a dictionary.

5.3. Evaluations

All experiments were conducted on an NVIDIA 1080Ti GPU and an Intel E5-2690 processor. We did not apply the early stopping support of Calamari since the number of training instances is rather limited. Instead, training was stopped after 20,000 iterations whereby the best performing model on the training data so far was chosen for evaluation. Following further hyperparameters were set: 0.001 as learning rate, the Adam solver [22] for the gradient descent, and a dropout [23] of 0.5 after the LSTM layer. Furthermore, we used the provided data augmentation of Calamari with a factor of 50, that is, each line is multiplied 50 times in addition to the original one.

5.3.1. Preliminary Experiments

This section presents and discusses the results obtained when varying the network architecture. For the experiments, we took all three parts of Nevers as dataset which we split into five cross-folds to train networks, each on four folds and evaluated on the remaining one ("All" schemata in Figure 6). The obtained average values of the hSAR, dSAR, and NAR metrics are listed in Table 2.

We focused on varying the kernel size and thus stride of the pooling layer because, in contrast to the ATR, NCs can be located very dense, even above of each other. It is important to maintain an output that is long enough to predict each neume: Without a pooling layer, a symbol can be predicted at each pixel position, a 2×2 pooling layer halves the dimension of the output which is why relative to the input image only every second pixel a symbol can be predicted. Unfortunately, max-pooling layers whose aim is to reduce the dimensions are a obligatory component of a CNN architecture because they keep only the important features of the previous layers and also introduce a rotation, scale, and

Table 3. Results (lower half) when training the proposed CNN/LSTM network on the three parts of the Nevers dataset using the "pretraining" schemata of Figure 6. For comparison, the FCN results of the upper half are obtained by using the methodology of [7]. All values are given in percent.

		hSAR				dSAR			
	N_{train}	1	2	3	Mean	1	2	3	Mean
FCN	0	93.6	85.2	90.7	89.8	86.7	80.5	87.5	84.9
	1	89.9	82.3	90.6	87.6	78.3	76.8	84.5	79.9
	2	90.5	84.8	89.3	88.2	83.9	79.8	86.0	83.2
	4	92.8	86.4	90.9	90.0	86.5	81.5	88.5	85.5
	All	93.6	91.4	93.4	92.8	88.6	87.9	90.7	89.0
CNN/LSTM	0	74.5	82.0	79.9	78.8	69.0	78.1	76.7	74.6
	1	84.2	84.0	83.6	84.0	75.7	80.4	79.7	78.6
	2	91.4	84.5	90.2	88.8	83.6	80.9	88.4	84.3
	4	92.1	85.5	92.5	90.0	85.3	80.9	90.6	85.6
	All	93.4	91.7	92.6	92.6	87.5	88.8	90.9	89.1

translation invariance of the features to a certain amount. Therefore, the different tested network architectures introduce max-pooling layers with a traditional 2×2 shape, but also a 1×2 shape which does only affect the vertical resolution but not the horizontal which is why it has no effect on the number of symbols that can theoretically be predicted by the network.

The network with the best results (NA4 in Table 2) showed that using a deep network architecture is superior to a network with only two convolutional layers, using 1×2 max-pooling layers. Furthermore, we conclude that an increased number of hidden nodes in the LSTM-layer leads to overfitting which is why the accuracy shrinks. Also, the usage of at least one 2×2 shape max-pooling layer is sensible. In all further experiments, we applied the network architecture of NA4 since it performs best.

5.3.2. Varying the Number of Training Examples

The number of freely available datasets of the target material to train the proposed neural network for the symbol detection is rather limited. Therefore, it is impossible to train and share a so-called mixed model that fits a variety of different books which is why book-specific models must be trained. Since the manual creation of GT is very tedious, it is mandatory to evaluate how many training instances are required in average to train a model of a certain accuracy. Naturally, the actual amount of GT highly depends on the material at hand.

In this section, we trained and evaluated the network using a fixed number of training instances (0, 1, 2, 4, and All) of one part and applying a pretrained model on the remaining ones (second setup in Figure 6). Using 0 pages for the actual training equates to only using the model that was pretrained on the remaining parts. Table 3 shows the results of the hSAR and dSAR for each part and each N_{train} and also the average across all parts. For comparison, we first added the results obtained when applying the FCN-based algorithm which we had proposed in [7], then the results of the CNN/LSTM-based approach are listed.

As expected the number of training instances increased the accuracies: Adding one page increased the average dSAR from 74.6% to 78.6%, a second one further improved the results to 84.3%. Finally, using all available training pages yielded a dSAR of 89.1%. The overall accuracies of part 2 were worse than of both other parts, which was expected since this part is the most difficult one. Similarly, the results of part 1 comprising the pages with the highest quality were a bit better than part 3. Interestingly, however, the pretrained CNN/LSTM ($N_{\text{train}} = 0$) performed best on the 2nd part and the worst on part 1.

Compared to the FCN-approach, the novel approach using CNN/LSTMs yielded slightly improved results of the dSAR if four or more pages of GT were available. The effect that one page of

Table 4. Incorporation of a dictionary for decoding. All values were obtained by training with N_{train} pages on one part and using a pretrained model trained on the remaining ones. All values are the result of a five cross-fold and given in percent. The last row lists the duration required to decode a single staff.

LM	**None**		**Token**		**Beam Search 25**		**Beam Search 100**	
N_{train}	dSAR	NAR	dSAR	NAR	dSAR	NAR	dSAR	NAR
0	74.6	67.5	75.0	68.7	75.1	69.0	75.3	69.3
1	78.6	72.1	78.9	73.1	78.8	73.2	79.0	73.2
2	84.3	77.0	84.5	77.9	84.5	78.0	84.6	78.1
4	85.6	78.4	85.9	79.3	85.8	79.3	85.8	79.3
All	89.1	82.7	89.2	83.5	89.1	83.5	89.1	83.5
Time	0.23 s		2.83 s		1.66 s		8.59 s	

training data reduces the accuracy of the pretrained net possibly due to overfitting did not emerge in our new approach. We think the reason is that the CNN/LSTM-network has to learn a more general representation compared to the FCN: Whereas the exact locations of notes are presented to the FCN, the novel network architecture has to learn by itself to concept of a note. This is also the reason why the approach was significantly worse if only a few lines of GT were available for training.

5.3.3. Incorporating a Neume Dictionary

To further improve the sequential prediction which requires a decoding of the probability matrix produced by the neural net, it is sensible to only allow for valid neumes. Thereto, we created a dictionary of all occurring neumes (see Section 3) and applied the presented decoding algorithms (see Section 4.2.2) which restricted the prediction to clefs, accidentals, and known neumes.

Table 4 shows the results of the dSAR and NAR when applying no dictionary, the CTC-TP, CTC-BS with $n_{\text{bw}} = 25$, and $n_{\text{bw}} = 100$, respectively. Hereby, we used the previous cross-fold-scheme when training with N_{train} images of one part and took the pretrained model of the remaining ones (second setup in Figure 6). Only the average values of all three parts are listed. Note, that the difference between the dSAR and NAR is considerably smaller compared to the analogous WAR in ATR because neumes are in average composed of considerably fewer symbols than words, some neumes even comprise a single note only. The reason is that it is more probable that a short neume containing an error matches with another one in the dictionary and can therefore not be corrected. Furthermore, the required decoding times for a single staff line are listed for each decoding approach.

As expected, the accuracies increased if incorporating a neume dictionary, however only to a very small amount (about 5%). When using only the pretrained models (0 lines), the NAR improved from 67.5% to 68.7% if the token passing decoder was chosen. The beam search decoder with $n_{\text{bw}} = 25$ yielded a sightly higher NAR of 69.0% and using 100 beams results, as expected, in the best value of 69.3%. If four or all training pages were used, all tree decoders with dictionary yielded approximately the same value of NAR $= 79.3\%$ and NAR $= 83.5\%$, respectively, which was higher than using no dictionary (78.4% and 82.7%).

A crucial difference of all examined decoding approaches was the duration required to process a single line which is listed in the last row of Table 4. Even the beam search algorithm with $n_{\text{bw}} = 25$ (1.66 s) was more than six times slower than using the default greedy decoder (0.23 s). The token passing decoder was even slower (2.83 s), and, as expected, the beam search decoder using 100 lines (9 s) was the slowest one. These observations are in accordance with the time complexity of the different decoders as shown in Section 4.2.2. The overall duration could be clearly reduced using a C++ implementation instead of the Python version of the algorithms, however, it is still up for debate if the cost-benefit ratio of the beam search algorithm with $n_{\text{bw}} = 100$ is sensible. Instead, the evaluation leads to the conclusion that using a beam search with only 25 beams (or even less) is completely sufficient and also acceptable in terms of duration.

Table 5. Evaluation of using the pretrained model on the Nevers dataset for the two other books Assisi and Cambrai. Since the Assisi dataset only comprises five pages, training on all pages equates to training on the four pages in the cross-fold.

Dataset	N_{train}	Default hSAR	Default dSAR	Pretrained hSAR	Pretrained dSAR
Nevers	Best	91.7	88.3	–	–
Assisi	0	–	–	94.6	92.0
	1	63.2	54.9	93.5	90.7
	2	83.3	76.7	94.0	91.0
	4/All	87.9	83.5	95.0	92.7
Cambrai	0	–	–	47.8	39.2
	1	33.0	26.8	60.3	53.8
	2	53.2	43.6	70.2	63.9
	4	69.7	61.8	82.4	77.4
	All	87.0	81.5	90.3	86.0

5.3.4. Cross-Dataset Training

In this section, we evaluate how well a model pretrained on the Nevers dataset performs on different books. Thereto, we first trained default models for the Assisi and Cambrai datasets without using any information from the Nevers dataset, then we repeated the experiments using the pretrained model on Nevers. Table 5 shows the obtained results.

If no pretrained model was used (default), the model trained on the Nevers dataset performed best with an dSAR of 88.3%. Even though only four pages could be used for training the Assisi dataset, its performance of dSAR $= 83.5\%$ was higher than training Cambrai with dSAR $= 81.5\%$. This confirms that Assisi is a very clean dataset while Cambrai suffers from high degradation and bleeding through the paper.

Using the pretrained model without training on the actual dataset ($N_{\text{train}} = 0$) shows that the notation style of Assisi is extremely close to the Nevers dataset because even without training a remarkable dSAR of 92.0% was reached which is even better than the result on the Nevers dataset itself (88.3%). The Cambrai dataset showed the opposite behaviour since the Nevers dataset only achieved a dSAR of 39.2%. Training on actual data for Cambrai was therefore important, however, all training instances of the cross-fold were required in order to obtain a model with a hSAR greater than 90.3%. In contrast, if training with one page of Assisi, the results worsened because the network overfitted on the one page and forgets the primary generalising weights. This is also caused by the training setup itself because we did not apply early stopping and chose the best model based on the training data. Four pages (All) were then required to actually benefit from the training data. All observations emphasised our initial observation of the poor quality of the Cambrai dataset.

5.3.5. Qualitative Evaluation of the Horizontal Note Position

Figure 7 shows the prediction of a page of the Assisi dataset using the best model which was trained on the four remaining pages with a pretrained model from the Nevers dataset. The centres of symbols are drawn at their predicted horizontal position, whereas the vertical position is computed based on the label of a symbol which is relative to the staff lines (for example the first red note in the first line is located in the 3$^{\text{rd}}$ space). A qualitative evaluation showed that the position of the notes (red boxes) was very accurate, however, with a small offset to the left. This confirms that the neural net correctly learned the semantic concept of a NC on its own, but produced the output as soon as a note starts and not in the centre of a NC. In contrast to the notes, all clefs that are the first symbol of a line had a large offset to their actual position even though the network seemed to learn their shape because, in most cases, the prediction whether a C or F clef is present was correctly distinguished (see e.g., the C-clefs in lines 2 to 5 and the F-clefs in the last two lines). A closer look at the predicted

Table 6. Categorisation of the different errors and their relative amount. See Figure 7 for examples.

	Group	Error
False Negatives		*53.5%*
1.	Missing Note	12.3%
2.	Wrong Connection	18.6%
3.	Wrong Location	12.6%
4.	Missing Clef	8.5%
5.	Missing Accidental	1.5%
False Positives		*46.5%*
6.	Additional Note	8.5%
2.	Wrong Connection	18.6%
3.	Wrong Location	12.6%
7.	Additional Clef	6.8%
8.	Additional Accidental	0.0%
	Total	100%

probability matrix shows that the network predicted the first clef always at the first possible location which is the start of the staff lines minus the padding. We think that the reason is, that the first symbol in a line is always a clef. Therefore, the network first learned to predict the most frequently occurring clef independent of the actual data. Later then, the actual concept of a clef and its type was learned, however, the location which was irrelevant in the loss-function stays the same. This is possible because the bidirectional LSTM allows to memorise the clef from the actual position the start of the line.

In general, apart from the initial clefs, the position of the symbols is close enough to the actual symbols in order to easily spot errors in an overlay view such as shown in Figure 7. Therefore, the algorithm can be integrated in existing semi-automatic OMR-frameworks such as OMMR4all [16] which already provides an overlay-editor for error correction.

5.4. Error Analysis

For a quantitative error analysis, we exemplarily took the best performing model (Assisi) and counted the number of insertions and deletions that were required to obtain the GT sequence. This approach counts replacements as two errors, one insertion and one deletion, which is why a wrong connection or a wrong vertical location was treated as one False Negative (FN) and one False Positive (FP). Table 6 lists the errors and their relative impact on the dSAR grouped in eight categories. Examples for the errors are shown in Figure 7, rare errors concerning accidentals are not present in the example.

First, as to be expected, there were clearly more missing notes than additional notes (12.3% vs 8.5%). The intrinsic reason is that the CTC-loss-function is rather restrictive in its prediction which is why blanks, that is no symbol, usually have a very high probability. The most probable errors were the prediction of a wrong connection with a relative amount of 37.2% in total (FP and FN). This is reasonable, because the decision if a note is, for example, the start of a neume or gapped can often only be decided based on context which in some occasions also includes the positions of syllables. Similarly, with a total relative error of 25.2%, the decision whether a note is located in a space or on a staff line also requires context, which is why it is not surprising that these kinds of errors occurred (see examples for the error type 3 "Wrong Location" in Figure 7). Even though, clefs usually occur only once or twice per line, they accounted for an error of 15.3% (error type 4 and 7), whereby missing clefs were more probable (8.5%, error type 4). Note, that the wrong offset of a clef at the start of a line did not count as an error. A qualitative evaluation showed that clefs were almost never predicted if they occur within a line, instead, they were often misinterpreted as notes. Accidentals are very rare which is why the network needed a very high confidence to predict at least some them. This explains why the amount of FPs-accidentals was 0.0%, and why missing accidentals were more likely. However,

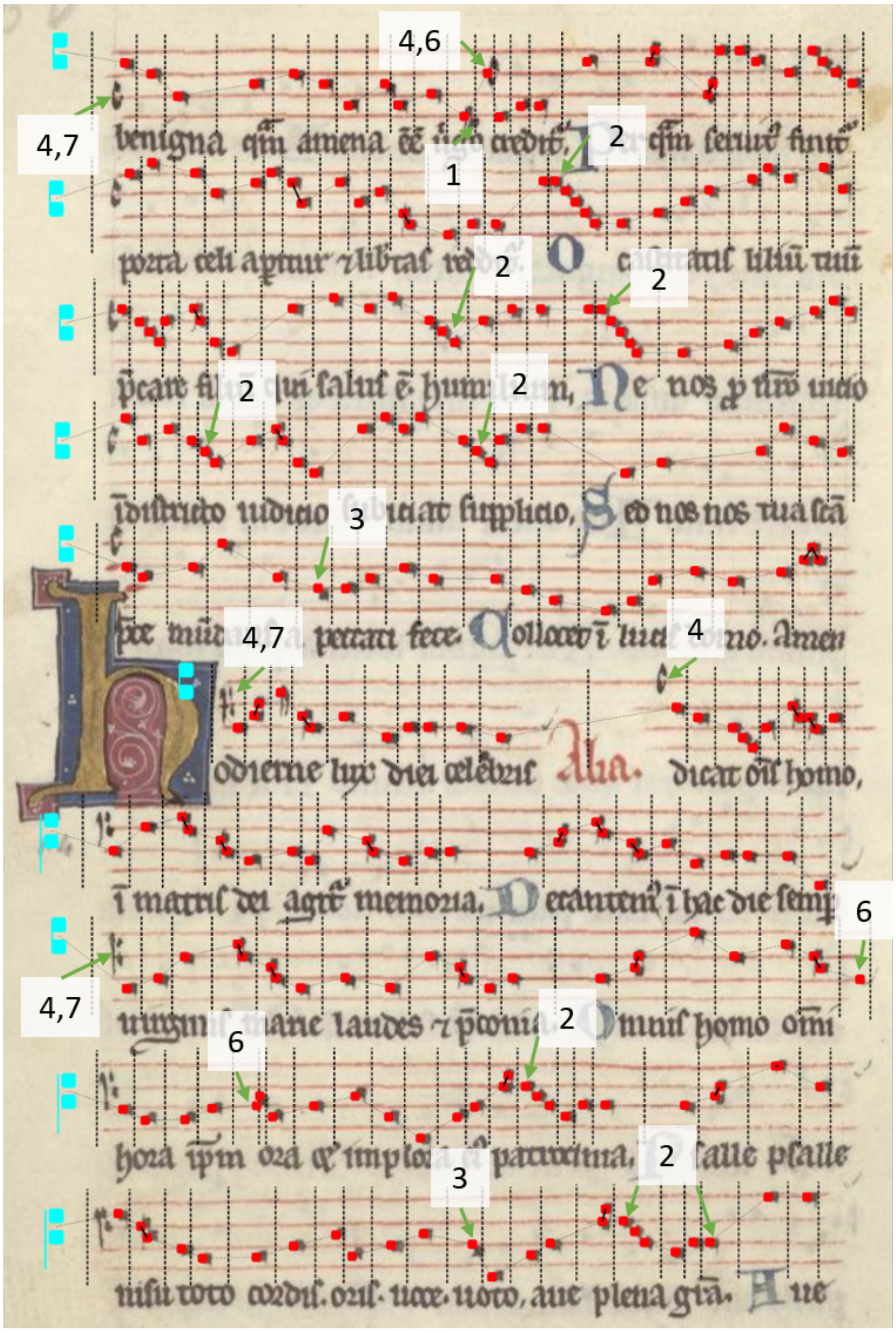
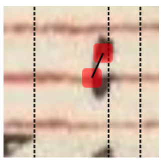
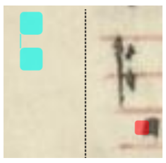
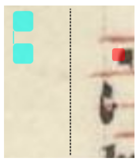
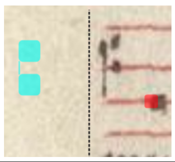


Figure 7. The full prediction of a page of the Assisi manuscript including the positional information: cyan symbols mark clefs, vertical dashed lines separate single neumes, NCs are rendered as red box, a looped graphical connection is indicated by a black stroke between to NCs. Arrows and the assigned numbers indicate the error type of the prediction (see Table 6). Note that the periphery of the page was manually cropped.

Table 7. Break down of the errors caused by clefs which amount for 15.3% of the total errors.

Error Type	Rel. Error	Examples
Within line Missing clefs that are not the first symbol of a staff line.	27%	
Beginning, location, and type Detected clefs at the beginning that have a wrong (vertical) location and type.	46%	
Beginning and location Detected clefs at the beginning that have a wrong (vertical) location, but a correct type	18%	
Beginning and type Detected clefs at the beginning that have a wrong type, but a correct (vertical) position.	9%	

the relative error of 1.5% caused by this type of errors is negligible if the lines are manually corrected anyway.

Next, we broke down the error causes for clefs into four categories (see Table 7): The first group which amounts for 27% of all clef-related errors counts all errors that are due to clefs that are located within the line (none of those were detected at all). All other three groups always refer to clefs that are the first symbol of a line: Clefs where both the location and type are wrong amount for 46% of the errors, clefs where only the location is wrong amount for 18%, and clefs with a wrong type amount for 9%. Positional errors (in total $46\% + 18\% = 64\%$) indicate that the network was not able to identify the clef at all, instead the network guessed using the background information that the first symbol is always a clef and that the C-clef location at the top line is the most probable one. This is supported by the fact that no clef was detected within a line at all. The least common error (wrong type only) shows that if the network was able to detect the correct location, its type is mainly correct. In conclusion, the error analysis shows, that the detection of clefs is still a challenging problem, presumably because they are underrepresented compared to notes. It is reasonable to incorporate the background information that the first symbol must be a clef to design a specialised algorithm to detect at least the first clef correctly. Ideally, this solves 73% of the clef-related errors, which would result in a relative improvement of the dSAR of about 10%.

Furthermore, we examined the FPs and FNs of the note detection to understand which sources could cause these kinds or errors (see Table 8). One third of all FPs are due to clefs or accidentals which generally look notes alike and that were falsely interpreted as clefs. One forth of the errors is caused by repeated notes, where the network predicted two NCs with a different location directly one after the other. This is an intrinsic problem if the CTC-algorithm, but can be solved in a postprocessing step that combines all notes that are very close (smaller than a typical note size) and are on the same location by choosing the most probable one. Another source of errors that can hardly be suppressed is noise, which amounts for 25% of the FPs. 17% of the errors occur at locations where NCs are written very densely, especially if the actual neume is a "stacked" one (e.g., pes or clivis). We expect that a postprocessing step can only reduce errors caused by repeated NCs, whereas any other source requires

Table 8. Break down of FP and FN errors into several causes which amout for 8.5% and 12.3% of the errors, respectively.

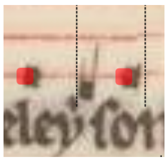
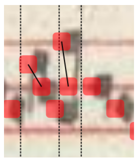
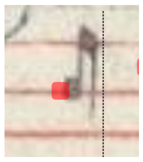
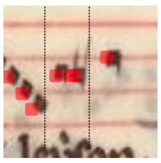
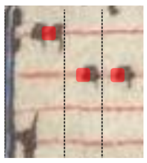
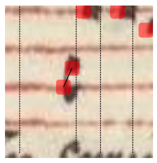
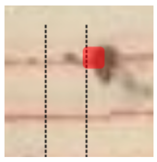
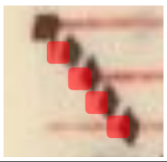
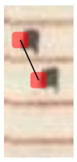
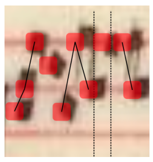
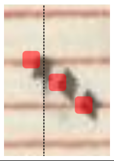
Error Type	FP	Examples	FN	Examples
Close to Lyrics (Bottom/Top) NCs that are likely to be misinterpreted as part of text.	0%		79%	
Stacked Neumes Errors caused related to NCs that are stacked or horizontally close to one another (e.g., pes, clivis).	17%		5%	
Repeated NCs The same notes are repeated.	25%		5%	
Clef/Accid A clef or an accidental is misinterpreted as notes.	33%		0%	
Noise Noise is falsely detected as NC.	25%		0%	
Other Other error sources that can not be directly explained.	0%		11%	

Table 9. Break down of the errors caused by wrong connections (start, looped, or gapped) which amount for 37.2% of the total errors.

Error Type	Rel. Error	Examples
Stacked Neumes Errors caused related to NCs that are stacked or horizontally close to one another (e.g., pes, clivis).	41%	 
Gapped Neumes Errors caused related to neumes with gapped NCs (e.g., climacus).	59%	

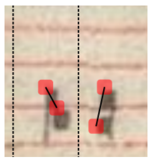
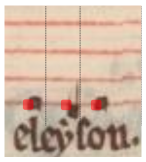
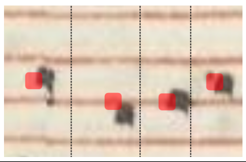
sophistication, such as additional context or more training material. Thus, ideally, about 25% of the FPs could be resolved which results in a relative improvement of the dSAR of about 2%.

The main source of FNs (79%) are NCs that lie close the top or bottom of a line. There, the NCs are very narrow to the lyrics which is an additional challenge for the network because it must also separate text and actual notes. We do not think that these kinds of errors can be easily postprocessed without additional training material that focuses the separation of text and music. For the other three types of errors, we have no explanation, why the network misses the NCs. It seems that there is an accumulation of errors if the NC is the first one in a line, though. Postprocessing algorithms to tackle some of the FN errors seem not feasible easily.

Next, we examine the origin of NCs with a correct location, but with a wrong connection which amount for 37.2% of the total dSAR error. The first category (41%) comprises errors that are related to two or more NCs that are stacked or very close in horizontal direction (e.g., pes or clivis). These situations are sometimes difficult to decide for the network because it has to recognise if notes are graphically connected, which can sometimes be very thin. Instead, we think that the errors are caused because the network uses only positional information to decide about the connection which is in most cases valid. This explains why the examples are wrongly classified, however a postprocessing step does not seem feasible to solve these kinds of errors. The second category (59%) comprises errors that occur in gapped neumes (e.g., a climacus). Here, the network must decide whether a note is the start of a new neume or part of the same previous neume. There are some cases where the lyric is required to decide about the connection which is why it is reasonable that the network makes these kinds of errors. In the first category, the incorporation of background knowledge to solve these kind of errors is difficult because it requires the detection of the actual connecting line. The incorporation of syllables is feasible, however only if the syllables and their positions are known. Therefore, for an actual realisation of both postprocessing steps, a second challenging problem must be solved first.

The last error source are positional errors of notes which amount for 25.2% of the errors (see Table 10). The errors are caused because notes that lie on the same vertical pixel position can be differently interpreted relative to the staff lines depending on the context. Therefore, we broke down the errors into the causes: stacked neumes (65%), close to lyrics (12%), and other (e.g., middle of the line, 23%). The error examples show, that a NC that touches a staff line is usually "on the line" if it is part of a stacked neume (e.g., pes or clivis), but in the space if it is a single tone neume. We think that these errors could be resolved by combining this approach either with our FCN approach (if valid GT is available) or with a note detection based on connected commponents in the area of interest. If the actual (vertical and horizontal) position of a NC is known, a set of rules could resolve the errors: In a stacked neumes, a position close to a staff line is interpreted as "on the line", else it is interpreted as "space". The threshold could be derived based on majority voting of all present notes. Ideally, this could reduce the dSAR by 25.2%, relatively.

Table 10. Break down of the errors caused by a wrong (vertical) location which amount for 25.2% of the total errors.

Error Type	Rel. Error	Examples
Stacked Neumes Errors caused related to NCs that are stacked or horizontally close to one another (e.g., pes, clivis).	65%	
Close to Lyrics (Bottom/Top) NCs that are likely to be misinterpreted as part of text.	12%	
Other Error causes without a specific category (e.g., single tone neumes in the middle of a line).	23%	

Concluding, the error analysis showed that there are two major error categories: Errors that require external knowledge (e.g., lyrics) and errors where the local information presented to the network should be sufficient (e.g., positional errors, or clefs). We expect, that errors of the last group could be (mostly) corrected by specialised postprocessing steps which could potentially reduce the dSAR by up to 37.2%.

6. Conclusion and Future Work

In this paper, we presented the application of CNN/LSTM-hybrids to tackle the automatic transcription of Medieval square notation. The approach reached a best value with a dSAR of 89.1% when training on the three parts of the Nevers dataset and is thus comparable to the previous value of 89.0% achieved by an FCN. For only a few pages of GT (zero to two), the FCN was clearly better, though. Concluding, if the target material consists of many pages, it is meaningful to apply the CNN/LSTM-based approach because it yields competitive results and allows for an easier GT-production. Using a neume dictionary for decoding increased the NAR of the best model from 82.7% to 83.5%. It shows that for many pages the fast beam search algorithm using $n_{\mathrm{bw}} = 25$ was sufficient.

Astonishingly, the pretrained model on the Nevers dataset yielded better results on Assisi with a dSAR of 92.0% than on the actual training data. Finetuning on Assisi increased the performance to 92.7%. In contrast, the pretrained model, but also actual training performed worse on the difficult Cambrai dataset with a dSAR up to 86%.

While the overall accuracy of the prediction of the horizontal position was completely sufficient, the error analysis showed that the first clef of each line is off because it was predicted directly as the first output of the network. We expect that data augmentation by prepending a section of another line might prevent this. This approach is only possible if positional information such as in the Nevers dataset is available, though.

The error analysis showed that the main sources of errors were NCs with a wrong connection or a wrong vertical location. This requires either more context or external knowledge, for example, about more or less probable melodies, which could be directly incorporated in the language model for neumes or NCs based on n-grams. However, to obtain reliable transition probabilities, several large datasets are required. Another problem to solve with the current CTC-TP- and CTC-BS-decoders is that they do not provide positional information required for the OMMR4all overlay-editor for post-correction. We plan to add support to both algorithms in order to allow for an actual productive inclusion of the language model.

Author Contributions: C. W. conceived and performed the experiments and created the GT data. C. W. designed the symbol detection algorithm. C. W. and F. P. analysed the results. C. W. wrote the paper with substantial contributions of F. P.

Funding: This research received no external funding

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ATR	Automatic Text Recognition
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
CTC-BS	CTC-Beam-Search
CTC-TP	CTC-Token-Passing
CWMN	Common Western Music Notation
FCN	Fully Convolutional Network
FP	False Positive
FN	False Negative
GT	Ground Truth
LSTM	Long Short Time Memory
NAR	Neume Accuracy Rate
NC	Note Component
NER	Neume Error Rate
OMR	Optical Music Recognition
RNN	Recurrent Neural Net
dSAR	diplomatic Symbol Accuracy Rate
dSER	diplomatic Symbol Error Rate
hSAR	harmonic Symbol Accuracy Rate
hSER	harmonic Symbol Error Rate
TP	True Positive
WAR	Word Accuracy Rate

References

1. Baroni, M.; Maguire, S.; Drabkin, W. The concept of musical grammar. *Music Analysis* **1983**, *2*, 175–208.
2. Grachten, M.; Arcos, J.L.; Lopez de Mantaras, R. Melodic similarity: Looking for a good abstraction level. 5th International Conference on Music Information Retrieval; , 2004.
3. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation* **1997**, *9*, 1735–1780.
4. Graves, A.; Fernández, S.; Gomez, F.; Schmidhuber, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. Proceedings of the 23rd international conference on Machine learning. ACM, 2006, pp. 369–376.
5. Breuel, T. High Performance Text Recognition Using a Hybrid Convolutional-LSTM Implementation. 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). IEEE, 2017, pp. 11–16.
6. Wick, C.; Reul, C.; Puppe, F. Comparison of OCR Accuracy on Early Printed Books using the Open Source Engines Calamari and OCRopus. *JLCL: Special Issue on Automatic Text and Layout Recognition* **2018**, *33*, 79–96.
7. Wick, C.; Hartelt, A.; Puppe, F. Staff, Symbol and Melody Detection of Medieval Manuscripts Written in Square Notation Using Deep Fully Convolutional Networks. *Applied Sciences* **2019**, *9*, 2646.
8. van der Wel, E.; Ullrich, K. Optical Music Recognition with Convolutional Sequence-to-Sequence Models. 18th International Society for Music Information Retrieval Conference; , 2017.
9. Calvo-Zaragoza, J.; Valero-Mas, J.J.; Pertusa, A. End-to-end Optical Music Recognition using Neural Networks. 18th International Society for Music Information Retrieval Conference; , 2017.
10. Pugin, L.; Zitellini, R.; Roland, P. Verovio: A library for Engraving MEI Music Notation into SVG. ISMIR, 2014, pp. 107–112.
11. Baró, A.; Riba, P.; Calvo-Zaragoza, J.; Fornés, A. From Optical Music Recognition to Handwritten Music Recognition: A baseline. *Pattern Recognition Letters* **2019**, *123*, 1–8. doi:https://doi.org/10.1016/j.patrec.2019.02.029.

12. Calvo-Zaragoza, J.; Hajič jr., J.; Pacha, A. Understanding Optical Music Recognition. *Computing Research Repository* **2019**.
13. Hajič jr., J.; Pecina, P. The MUSCIMA++ Dataset for Handwritten Optical Music Recognition. 14th International Conference on Document Analysis and Recognition; , 2017; pp. 39–46. doi:10.1109/ICDAR.2017.16.
14. Ramirez, C.; Ohya, J. Automatic Recognition of Square Notation Symbols in Western Plainchant Manuscripts. *Journal of New Music Research* **2014**, *43*, 390–399. doi:10.1080/09298215.2014.931438.
15. Calvo-Zaragoza, J.; Toselli, A.; Vidal, E. Handwritten Music Recognition for Mensural Notation: Formulation, Data and Baseline Results. 14th International Conference on Document Analysis and Recognition; , 2017; pp. 1081–1086. doi:10.1109/ICDAR.2017.179.
16. Wick, C.; Puppe, F. OMMR4all — a Semiautomatic Online Editor for Medieval Music Notations. 2nd International Workshop on Reading Music Systems; Calvo-Zaragoza, J.; Pacha, A., Eds.; , 2019; pp. 31–34.
17. Wick, C.; Reul, C.; Puppe, F. Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition. *Digital Humanities Quarterly (forthcoming)* **2019**.
18. Eipert, T.; Herrman, F.; Wick, C.; Puppe, F.; Haug, A. Editor Support for Digital Editions of Medieval Monophonic Music. 2nd International Workshop on Reading Music Systems; Calvo-Zaragoza, J.; Pacha, A., Eds.; , 2019; pp. 4–7.
19. Graves, A. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*; Springer, 2012; pp. 5–13.
20. Scheidl, H.; Fiel, S.; Sablatnig, R. Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm. 16th International Conference on Frontiers in Handwriting Recognition. IEEE, 2018, pp. 253–258.
21. Hwang, K.; Sung, W. Character-level incremental speech recognition with recurrent neural networks. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016, p. 5335–5339. doi:10.1109/ICASSP.2016.7472696.
22. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *CoRR* **2014**, *abs/1412.6980*.
23. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* **2014**, *15*, 1929–1958.