From image to MIDI: Implementing a complete OMR system for sheet music

Arnau Josep Alcon Acedo

Resum—Llegir de manera computacional la informació que conté una partitura presenta molts reptes. Un d'aquests reptes és la necessitat de coordinar diversos processos diferents. Aplicar deep learning permet consolidar alguns d'aquests processos en un sol pas. Aquest projecte proposa un sistema complert de predicció de partitures partint d'un model seq2seq que pren petits troçets de partitura, de la mida d'un compàs, en format PNG a l'entrada, i crea un arxiu MIDI amb la predicció a la sortida. S'implementen les funcionalitats necessàries per tenir el pipeline sencer. Aquestes funcionalitats inclouen segmentar la partitura en compassos, realitzar prediccions sobre aquests, i agrupar tots els MIDIs resultants en un de sol.

Paraules clau—OMR, segmentació de partitures, MIDI (Musical Instrument Digital Interface), pentagrames, compassos, deep learning, seq2seq

Abstract—Computationally reading the information contained in music scores presents many challenges. One of those challenges is having to coordinate several separate processes. Deep learning allows to consolidate a few of these processes into a single step. This project proposes a complete sheet detection system with a seq2seq solution as its base that takes small, bar-sized sheet music chunks in PNG format at the input and creates a prediction MIDI file at the output. It implements the required functionalities to have the entire pipeline. These functionalities include segmenting a score into its bars, doing predictions on these segments and combining all the MIDI outputs into a single MIDI file.

Index Terms—OMR, score segmentation, MIDI (Musical Instrument Digital Interface), staves, bars, deep learning, seq2seq

1 INTRODUCTION -

Optical Music Recognition (OMR) is the field of research that studies how to computationally read musical notation in documents, as defined by Calvo-Zaragoza et al. [1]. The goal is to teach a computer to identify, read, interpret and transcribe the information contained in sheet music into a format the machine can understand.

This field of study shares many similarities with the field of Optical Character Recognition (OCR), whose goal is to read and interpret typed, handwritten, or printed text instead. During the last 20 years, however, OCR has seen significantly more advances than OMR.

Calvo-Zaragoza et al. [1] identify and list numerous factors that contributed to slowing down research in this field. Notable factors include the complex nature of the information conveyed through music notation, its 2-Dimensional structure, its inconsistent representation specifically in older documents, and the lack of a unified and cohesive research.

In recent years, however, OMR has seen a resurgence with the advancements in deep learning, which have shown very promising results in contrast with older OMR solutions. However, as put by Bellini et al. [2] "despite the availability of several commercial OMRs: PIANOSCAN, NOTESCAN in Nithingale, MIDISCAN in FINALE, PohotoScore in Sibelius, SmartScore, SharpEye, etc., none of these is satisfactory in terms of precision and reliability. They provide a real efficiency close far from the 100% only when quite regular music sheets are processed. This justifies the research works around the definition of reliable OMR algorithms".

One such algorithm is the one proposed by Baró et al.[3], Riba et al.[4] and modified by Santaella [5]. They implement an OMR solution based on a seq2seq model. This solution has been trained on small, bar-sized sheet music chunks in PNG format synthetically generated by a script. It has shown good results when tested against other PNGs generated by said script.

This project implements the necessary steps to, using this algorithm as its basis, be able to perform predictions on a full page of sheet music.

• *Contact E-mail: arnaujalcon@gmail.com*
• *Menció realitzada: Enginyeria de Computació*
• *Treball tutoritzat per: Alicia Fornes Bisquerra (Computer Science)*
• *Academic year 2022/23*

2 Objectives

The main goal of this project is creating a program that takes a PNG of a music score as an input, does the necessary processing to transform it into the preferred inputs of the seq2seq model, executes a prediction on all these inputs, collects all of the MIDI outputs and combines them into a single MIDI file representing the information read from the original PNG.

The objectives the project needs to accomplish to achieve this goal are the following:

- To research and understand the implementation of the seq2seq model, and understand the formats it uses as inputs and outpus.
- To implement functionalities that segment a PNG of a music score into the small, bar-sized chunks the seq2seq algorithm works best with.
- To implement functionalities that assemble the output of executing the algorithm on the segmented bars into a single MIDI file.
- To research the state of the art of OMR.
- To automate the connections between these functionalities in a pipeline, from full sheet PNG to MIDI.
- To test the accuracy of this system's readings with real music scores to validate its practical application.

3 State of the art

Most of the work in the OMR field can be found in scientific publications each tackling different aspects of the process. Although commercial applications do exist, there does not exist a leading entity that defines a main general implementation.

Most of the existing software does follow a similar pipeline that involves image pre-processing, symbol recognition and information reconstruction [6]. The techniques used in every step do vary from project to project, and there exists a lot of research exploring different techniques and implementations.

Due to the nature of the information conveyed in music scores, combined with the complexity of its representation, older symbol recognition solutions involved a lot of steps, each tailor-made to tackle a specific part of the process. This is a sensible approach considering computers work best when tackling specific and well-defined problems. However, chaining all these processes together resulted in their error rates, which were often acceptable for each individual step, multiplying together to unacceptable amounts for what is expected of a practical application.

The recent development of deep learning technologies has allowed for a new way of tackling this field. Deep neural networks have demonstrated an ability to tackle generalistic problems very broad in scope. A lot of recent research focuses on exploring different OMR solutions involving deep neural networks, following in the steps of its brother field, OCR, which is seeing huge advances thanks to these.

The algorithm that this project adopts uses the neural network model known as sequence-to-sequence, or seq2seq in short, originally proposed by Mikolov in his PhD thesis [7]. This family of machine learning approaches' primary use is natural language processing. As the name implies, seq2seq models turn one sequence into another sequence. What's particular about them is that they use the previous output as the input context.

Unfortunately, not many databases are publicly available for OMR related work. Deep learning models' accuracy often scales with database size, which severely hurts the results of a lot of smaller scale research. The seq2seq has been trained on synthetically generated sheet chunks, which made it possible to work with a decent sized database, but in exchange has performance issues when the sheets style and font differs too much from the generated ones.

Regarding image pre-processing, this paper by Vinaya V et al. [8] touches on staff line removal, and the technique they use to find the regions where the lines are will be useful in this project alongside other techniques for line and bar segmentation. I have not found any work that uses convolutions on custom kernels for staff line removal, a technique that I thought could be useful and will use in this project.

Finally, regarding the step of information reconstruction, the MIDI reconstruction that I do in this project is not really a part of the OMR field of study. MIDI is an old format with extensive documentation. The educational content creator javidx9 [9], also known as OneLoneCoder, has a publicly available friendly implementation of a MIDI parser [10] that has been very helpful in understanding the MIDI specification and how I should approach my task.

4 Methodology

4.1 Pipeline structure

My system operates on three main blocks, which are:

1. Music score segmentation
2. Seq2seq prediction
3. MIDI assembly

The first block takes the PNG of a page of sheet music and segments it into lines, each of them is then segmented into bars, and these bars are temporarily saved as independent PNGs in a separate folder.

The second block takes these PNGs one by one and executes a prediction using the trained seq2seq model, outputting that prediction as a MIDI file for each PNG.

The third block parses all these MIDI files, records the noteON and noteOFF events it finds, which are the sections of the file relevant to playing back the song, and stitches them all together alongside the necessary headers and meta events, outputting a single MIDI file with the information read from the original sheet music.

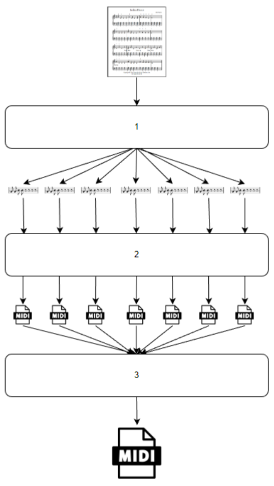


Figure 1: Diagram of the project's pipeline

4.2 First block: Music score segmentation

This block takes a PNG image at its input. It is assumed that the image is of one page sheet music. Its goal is to segment this image into the small, bar-sized chunks that our model can make predictions on. These segments do not need to correspond to the actual bars in the original sheet, but segmenting the bars is a way to do it. The segments of the image are then saved independently in a separate folder to feed into the seq2seq later.

4.2.1 Image pre-processing

Before segmentation starts, the image gets resized to 500 pixels wide per 600 pixels high. Most PNGs of a music score page are around this size, which is why I settled on these values. This is to prevent unexpected outcomes caused by inputting an excessively large or small PNG, while minimazing the impact the resizing has on the type of image the algorithm expects.

Depending on how the score image was taken it may present pixels that do not belong to the sheet around the edges that could be misinterpreted in line or bar detecting, so we crop the image's edges. We apply a four-point transform to straighten the image up in case it was slightly rotated, and we grayscale and denoise it as well.

4.2.2 Line segmentation

Music scores almost always consist of several lines. Each line is read from left to right and contains relevant information on what the musician must play at a certain point in time in the song. In a similar way to written text, when all the information in a line has been read, the line directly below it contains the information that should be read next.

Line segmentation is the process of dividing the music sheet with straight, horizontal lines spanning the entire width of the sheet such as that the regions delimited by these lines represent a music line each.

The preferred method for line detection is doing a horitzontal projection of the image. The image is binarized and for each of the 600 pixels of height, the binary value of the 500 pixels at that height is added up. This results in a one-dimensional array of size 600 where each value corresponds to the amount of non-zero pixels in each row. The lines having the staff will result in high peaks. The position of these peaks in the histogram gives the location of staff lines.

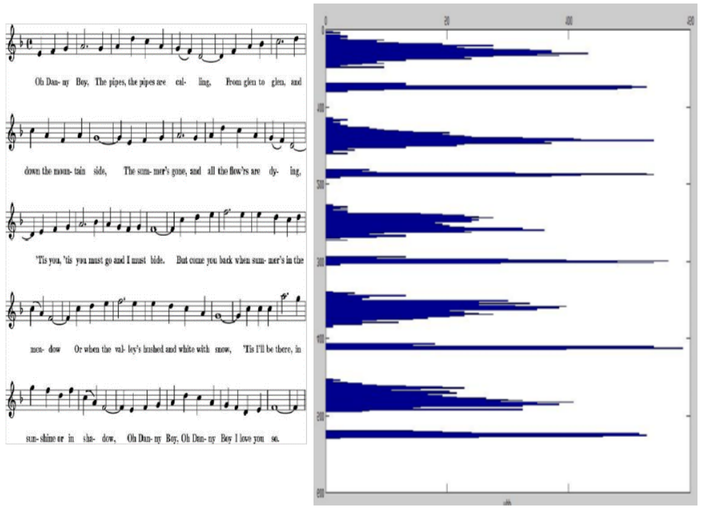


Figure 2: Input image and corresponding output histogram

When I started investigating line segmentation I expected the use of convolutions with custom kernels and was surprised to find no work with that particular approach. I wanted to try a second approach to line segmentation using this particular technology and compare their performances.

Convolutions are a very potent tool in the field of computer vision. Convolving an image with a small matrix called kernel results in a new image. Each pixel in that new image holds a relation to the same pixel in the original image and the pixels around it. What that relation is depends on the kernel used. Common applications are image filters, blurring and edge-detection. Convolutions are a pretty deep and interesting topic but I won't get into the details in this paper. For a better, more extensive explanation I highly recommend 3Blue1Brown [11]'s video on the subject.

For convolutions, I used the SciPy library's implementation using FFTs (Fast Fourier Transform), which is essentially a much quicker and less computationally expensive way to calculate a convolution. My approach convolutes the whole music sheet input with a custom-made kernel that highlights in white the pixels that are part of horizontal lines and paints black those that are not. This results in notes being erased while the overall staff outline remains and is highlighted. But we only care about detecting lines, not staffs.

The resulting image is convoluted with a second custom kernel, that consolidates white regions into large white clusters. This results in a third image where all the information of the original sheet has disappeared except for the regions spanning each line. A single iteration of cv2 library's dilation convolution is applied to eliminate any remaining noise and further consolidate the white areas.

Now we can very easily search lines around this image's central area. A very simplistic approach to this is iterating the 375th column of the image pixel by pixel to search for the regions of white pixels, the sheets' staves, and the regions of black pixels that separate them, and returning the height at the centre of each black region. Drawing a horizontal line at each of these heights on the original sheet results in it being segmented by lines.

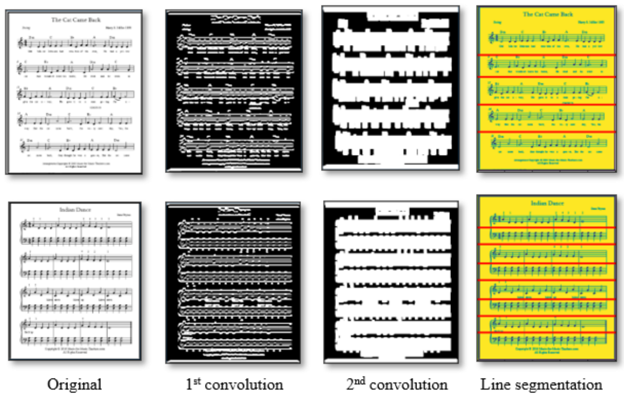


Figure 3: Single staff and dual stave sheets at different stages of segmentation

The reason I arbitrarily selected the 375th column (which represents the 3rd quarter of the image of a total width of 500) is because the right-most and left-most edges of the sheet do not contain staff lines, and the first quarter and center of the image often have titles that can obstruct the search. This means that the rare sheet with a title on the third quarter can mess with this simple search, and a more advanced search method to circumvent this will be discussed on another section.

4.2.3 Bar segmentation

The goal of this step is to end up with smaller subdivisions of the original sheet that are better suited for predictions on our model. In theory, taking each line and dividing it in five equally sized chunks is a valid approach. This, however, risks cutting straight through a note, so using the bar divisions already present in the sheet is desirable.

Lines in sheet music are divided in several bars by straight, vertical lines spanning the height of the staff or staves in the line.

We want to find these vertical line divisions and segment the sheet using their positions. Santaella [3] implements a bar segmentation that involves using staff removal to erase horizontal lines from the sheet and then computes a vertical projection of the image. This is the same as the horizontal projection explained for line segmentation but for the other axis and executed on each segmented line. The image is binarized and for each of the 500 pixels of width, the binary value of the pixels at that widht is added up.

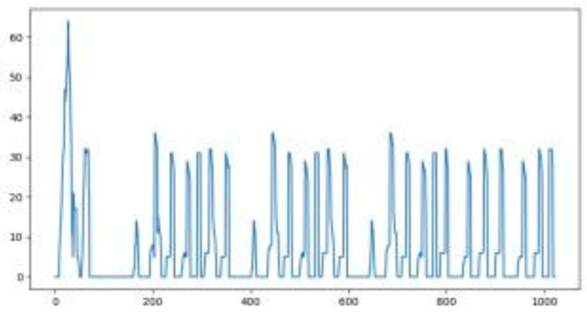


Figure 4: Horizontal projection of a staff

As Santaella [5] outlines in his paper, finding the bar divisions in this projection is much harder than it was finding the lines. He approaches the problem by searching for low variance points, since the bar divisions should all have about the same pixel height. I used his implementation of this approach.

I implemented a second approach to bar segmentation using a very similar system to the one I talked about in line segmentation. I convolute the whole music sheet input with a custom-made kernel that highlights in white the pixels that are part of vertical lines and paints black those that are not. This results in the exact opposite of the first convolution I do on line segmentation. Staff lines are erased and notes and bar divisions remain.

The resulting image is convoluted with a second custom kernel, that consolidates white regions into large white clusters. A single iteration of cv2 library's dilation convolution is applied to eliminate any remaining noise and further consolidate the white areas.

Now we can search for bar divisions by moving horizontally at each line's center and finding the white columns separating the black regions. This method, however, doesn't work for single-staff sheets, since these regions between staves in the same line aren't present there.

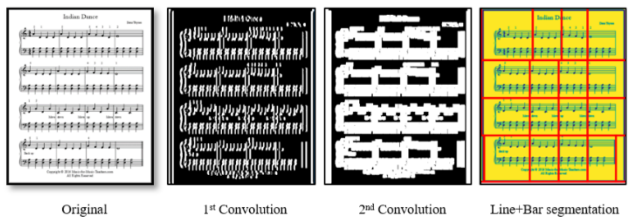


Figure 5: Dual stave sheet at different stages of segmentation

4.2.4 Voting system

The systems described in line segmentation and bar segmentation share a last step in which a single, arbitrary row/column of pixels is iterated in search of division lines.

This hypersimplistic approach works on visually clear, ideal scenario scores. However, its ability to detect divisions drops when executed on sheets with more visual noise. This issue can be mitigated by using a voting system.

Instead of iterating a single, arbitrary line of pixels, we iterate several, evenly spaced lines (columns for line segmentation, rows for bar segmentation). For each string, when a line or bar division is detected, the position of that pixel (height for line segmentation, width for bar segmentation) is added to a list just as I did in the simpler implementation. However, if this position has already been inserted into the list, it is instead added a "vote". When all the lines have been searched, the values of the list that have not reached a minimum number of votes are rejected, effectively removing most of the false detections caused by noise. When searching the pixel lines, before adding the position to the list, we round it to the nearest multiple of 10 to ensure that the actual divisions receive votes and are not rejected accidentally.

4.2.5 Mixed approach

I have also implemented a fourth approach to line and bar segmentation that combines the two ideas. It uses the same convolutions to highlight the relevant features of the image and then computes the horizontal and vertical projections respectively to search for the divisions. I believed they could compliment each other well.

4.3 Second block: Seq2seq prediction

This block executes predictions on all the images saved in a particular folder by the previous block, based on the trained model.

4.3.1 Seq2seq model

Sequence to sequence (seq2seq) is a family of machine learning approaches that turns a sequence, often a text string, into another sequence.

When trying to tackle problems such as language translation or text prediction using conventional neural networks, also called vanilla, it was noticed that the expected output did not depend solely on the input sequence, it was also dependant on the context around the input. This is an intrinsec characteristic of language. When we translate a sentence from one language to another, replacing every word in the sentence with its equivalent in the other language rarely results in a decent translation. Transmiting the underlying context of the sentence that we subconsciously understand is the key to a good translation.

One of the first successful solutions to this were Recurrent Neural Networks. RNNs output sequence is predicted using the input sequence and the previous output. This allows the network to have a memory of sorts, which uses to contextualize its predictions.

RNNs have a short memory though, and relevant context to the prediction that is found far away from the current sequence can be missed. This is known as the problem of vanishing gradient. LSTMs and GRUs are networks that solve this problem by judging the relevance of the context and forgetting what is less impactful. Phi's illustrated guide to LSTMs and GRUs [12] offers a deeper explanation.

The reason I explained all of this is because seq2seq makes use of an RNN, or more often an LSTM or GRU. It is an encoder-decoder model, which means its primary components are one enconder and one decoder network. The encoder turns each item into a corresponding hidden vector containing the item and its context, and the decoder turns the vector into an output item using the previous output as input context.

This project borrows the seq2seq model implemented and trained by Baró et al [3] and Torras et al [4], and modified by Santaella [5], to execute predictions. The model I will use has the following structure:

For the encoder, data is first introduced into a convolutional deep neural network (VGG19), then goes through a dropout layer and lastly is entered to a GRU.

For the decoder, the attention mechanism used is a combination of Bahdanau attention and location sensitive attention.

4.4 Third block: MIDI Assembly

This block takes the MIDI files saved in a particular folder by the previous block and outputs a single MIDI file that combines them in order.

4.4.1 MIDI file structure

The MIDI protocol was first released in 1983. It has managed to stay relevant for so long due to its clever design, it has adapted very well to technology's progress. But its flexibility comes at a cost. Parsing a MIDI file to extract specific information for computer processing is no easy task.

A MIDI file consists on a fixed size header followed by tracks. The header informs you of how many tracks there are in the file. Each track also has a header which informs about its size, and is followed by the MIDI events in that track.

MIDI events, or events for short, are the core of any MIDI file. This is where the complexity of parsing MIDIs is found. Since the protocol was created with the objective to maximize bandwith optimization and flexibility, the events data size is always contextual. For example, all events start with their Delta value. This value represents the time that should pass between this event and the previous one. This value can always be read onto a 4 byte integer. But in a MIDI file this value can be read from the seven less significant bits in 1 to 4 bytes, depending on whether each of those bytes has its most significant bit set or not. There are a lot more considerations that need to be accounted for when parsing a MIDI file that I will not get into here, more detailed explanations can be found in this educational video [13] and in the official specification [14].

In this project the two types of event that we care about extracting from the files the seq2seq outputs are NoteOn and NoteOff events. The MIDI protocol does not encode musical information in the way humans usually think about it, but rather how instruments use it. For instance, a keyboard that can communicate in MIDI protocol would create a NoteOn event whenever a key is pressed, and a NoteOff event when that key is released.

4.4.2 MIDI reading

I have repurposed OneLoneCoder's MIDI parser implementation in C++ [10] so that it ignores everything but NoteOn and NoteOff events. Whenever either of these events happens, the entire event is written to a separate text file that will be used later for reconstruction. I would like to stress that the flexible nature of MIDI events meant that implementing this reading process was far from simple.

An additional consideration was made to add a tick to the delta value of the first NoteOn event of every file except the very first one. This is because the first NoteOn event of a file starts at delta = 0, but following events that would happen at the same time are offset by 1 tick, an offset that the separated MIDI files do not account for.

4.4.3 MIDI construction

I have implemented a C++ class FuseMidis to construct my final MIDI file. You can keep adding as many MIDI files as you want through the method addMidi, that reads the NoteOn and NoteOff events and writes them onto a separate text file. When the method constructPrediction is called, the final MIDI file is crafted by stitching together a default MIDI header, a single track whose track length depends on the number of note events read, some predetermined track settings such as instrument, tempo and key signature, the note events read, and an end of track meta event.

5 Results

5.1 Line segmentation

I decided to evaluate line segmentation in binary terms: either the algorithm has segmented all the lines in the sheet correctly or it has not.

There are several factors that made me consider this evaluation scheme. The first is that in the context of this project, missing a single line or missing all of them is an equal degree of failure, since it will completely mess up bar segmentation and in consequence also prediction. The second is that detecting all lines in a sheet correctly is not too high a standard to expect of these algorithms. The third is that I have not seen a single case in my testing where an algorithm got all lines right except one. If the sheet's noise makes the algorithm miss a line, it often misses most of them and places lines where there should be none. The last consideration is that my testing dataset is small enough that I can afford to manually check the segmenting test results, and I did not believe that an automated a system to check the "correctness" of the line detection was worth implementing.

I tested 4 implementations of line segmenting: horizontal projection (P), convolution with simple detection (CS), convolution with voting system (CV) and convolution with horizontal projection (CP).

I tested them on databases containing 10 to 20 sheets of the following characteristics:

1. Ideal – The perfect sheet. High quality, evenly spaced, no title or lyrics, no noise.
2. Standard – A realistic good quality sheet. Some noise, finger notations, title.
3. Hard – A realistic bad quality sheet. Scribbled, poorly captured, a lot of noise.
4. Odd – Sheets with uncommon features. Lyrics, weird note fonts, few space between lines.

The results shown are the percentage of properly segmented sheets on each database.

	P	CS	CV	CP
1	100	100	100	100
2	100	43.75	100	100
3	76.92	30.76	69.23	84.61
4	60	0	20	60

Table 1: Line segmentation results

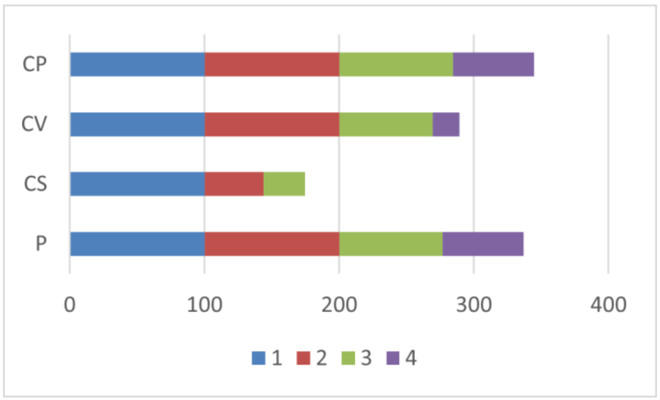


Figure 6: Stacked bar chart of line segmentation results

We observe that on ideal sheets all 4 algorithms detected lines without any issue. The simple detection's performance drops as soon as noise gets introduced, since something going wrong in any pixel of the arbitrarily chosen column/row makes the detection malfunction.

The voting system, as intended, makes the detection more resilient to noise. Its performance, however, also drops when the odd sheets are introduced. I have observed that this is almost always because larger noise clusters are not rejected by the voting system, resulting in line detections where there should be none. This problem could be fixed by either tweaking the voting system to be stricter or introducing a filter at the output that rejects lines that create too thin regions.

The horizontal projection and the convolution with horizontal projection perform about the same, which was to be expected. The convolution version has performed a bit better overall but my amount of testing is too low to determine that this approach is strictly better.

5.2 Bar segmentation

I evaluated bar segmentation with the same system I used to evaluate line segmentation for the same considerations.

I tested 4 implementations: vertical projection (P), convolution with simple detection (CS), convolution with voting system (CV) and convolution with vertical projection (CP).

I tested them on databases containing 10 to 20 sheets of the following characteristics:

1. Ideal – The perfect sheet. High quality, evenly spaced, no title or lyrics, no noise.
2. Standard – A realistic good quality sheet. Some noise, finger notations, title.
3. Hard – A realistic bad quality sheet. Scribbled, poorly captured, a lot of noise.
4. Odd – Sheets with uncommon features. Lyrics, weird note fonts, few space between lines.

All testing was done on lines properly segmented from the database sheets. The results shown are the percentage of properly segmented lines (not entire sheets) in each database.

	P	CS	CV	CP
1	100	87.5	100	100
2	90	37.5	100	100
3	41.67	27.78	47.22	75
4	43.33	0	16.67	60

Table 2: Bar segmentation results

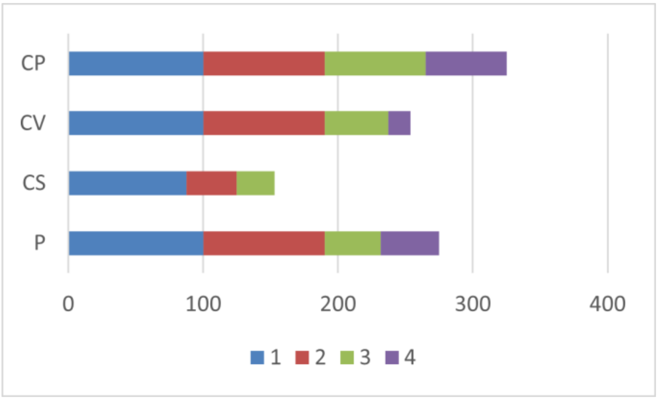


Figure 7: Stacked bar chart of bar segmentation results

The performance of the bar segmentation is worse overall than that of line segmentation.

The results follow the same pattern as the line segmentation, the ideal sheets are segmented almost perfectly, the simple detection's performance drops as soon as noise gets introduced, the voting system mitigates that problem and actually slightly outperforms the vertical projection on the hard database.

The convolution with vertical projection outperformed the vertical projection again but by a more meaningful margin this time, which leads me to believe that there might be some value to this approach.

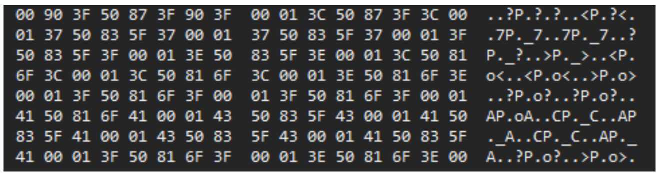
5.3 Seq2seq prediction

Santaella [5] already tested the performance of the seq2seq model on synthetic sheets and, at the time of writing, I have not yet been able to find a good way to evaluate its performance numerically.

That said, what predictions I have tested and checked manually so far have been relatively underwhelming. Even predictions on simple, ideal sheets that have been manually segmented are often off by a lot, though it does get some notes right every now and then so the original sheet is at least recognizible.

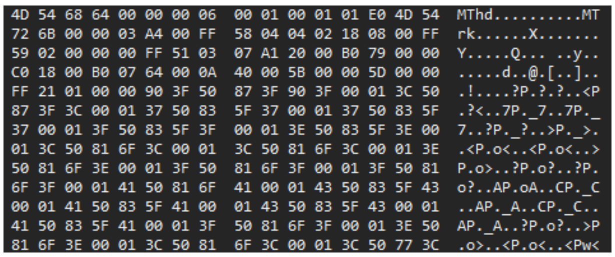
5.4 MIDI assembly

The MIDI construction has perfect performance. I have ensured that the program parses any and all MIDI files properly so long as they follow the specification. The note events are read correctly, and the final file is constructed with the exact notes read, in the correct order so long as the files were properly named for sorting, which in this pipeline they are.



```
00 90 3F 50 87 3F 90 3F  00 01 3C 50 87 3F 3C 00  ..?P.?.?..<P.?<.
01 37 50 83 5F 37 00 01  37 50 83 5F 37 00 01 3F  .7P._7..7P._7..?
50 83 5F 3F 00 01 3E 50  83 5F 3E 00 01 3C 50 81  P._?..>P._>..<P.
6F 3C 00 01 3C 50 81 6F  3C 00 01 3E 50 81 6F 3E  o<..<P.o<..>P.o>
00 01 3F 50 81 6F 3F 00  01 3F 50 81 6F 3F 00 01  ..?P.o?..?P.o?..
41 50 81 6F 41 00 01 43  50 83 5F 43 00 01 41 50  AP.oA..CP._C..AP
83 5F 41 00 01 43 50 83  5F 43 00 01 41 50 83 5F  ._A..CP._C..AP._
41 00 01 3F 50 81 6F 3F  00 01 3E 50 81 6F 3E 00  A..?P.o?..>P.o>.
```

Figure 8: Read notes text file, in binary editor



```
4D 54 68 64 00 00 00 06  00 01 00 01 01 E0 4D 54  MThd..........MT
72 6B 00 00 03 A4 00 FF  58 04 04 02 18 08 00 FF  rk......X.......
59 02 00 00 00 FF 51 03  07 A1 20 00 B0 79 00 00  Y.....Q... ..y..
C0 18 00 B0 07 64 00 0A  40 00 5B 00 00 5D 00 00  .....d..@.[..]..
FF 21 01 00 00 90 3F 50  87 3F 90 3F 00 01 3C 50  .!....?P.?.?..<P
87 3F 3C 00 01 37 50 83  5F 37 00 01 37 50 83 5F  .?<..7P._7..7P._
37 00 01 3F 50 83 5F 3F  00 01 3E 50 83 5F 3E 00  7..?P._?..>P._>.
01 3C 50 81 6F 3C 00 01  3C 50 81 6F 3C 00 01 3E  .<P.o<..<P.o<..>
50 81 6F 3E 00 01 3F 50  81 6F 3F 00 01 3F 50 81  P.o>..?P.o?..?P.
6F 3F 00 01 41 50 81 6F  41 00 01 43 50 83 5F 43  o?..AP.oA..CP._C
00 01 41 50 83 5F 41 00  01 43 50 83 5F 43 00 01  ..AP._A..CP._C..
41 50 83 5F 41 00 01 3F  50 81 6F 3F 00 01 3E 50  AP._A..?P.o?..>P
81 6F 3E 00 01 3C 50 81  6F 3C 00 01 3C 50 77 3C  .o>..<P.o<..<Pw<
```

Figure 9: The same notes in the assembled MIDI, in binary editor

There is an improvement I could make to this step. That is, the settings portion of the file is currently static and set to default values. For the purposes of this project this works perfectly fine, but if I want the file to play a different instrument or at another tempo I have to manually edit the settings text file. Having knowledge of the MIDI protocol this is not a problem for me personally, but having a more user-friendly way to select a few settings and those being automatically transcribed to the settings file would be a nice upgrade.

6 Conclusion

In this project I set off to implement a complete OMR system, one that spanned the entire pipeline from a complete music sheet PNG to a MIDI file. I have successfully achieved this goal.

Regarding score segmentation, I theorized a way to use convolutions to highlight the important features for line and bar detection, implemented it alongside the current standard, tested them both and found that combining both approaches has a real potential to be a better option than the current standard. More focused testing on this particular matter is needed to reach a definitive conclusion, but I believe it to be promising.

Regarding the predictions themselves, I'm not satisfied with the fact that I couldn't achieve a more thorough and scientific testing beyond manually executing and judging the results, but Santaella's previous work should be sufficient for now.

Regarding the MIDI file parsing, although my work on it is far from novel, I have had a lot of fun unravelling all the quirks of the protocol and I find it very clever and inspiring. I look forward to working with the format again in the future.

Finally, when it comes to the performance of the system as a whole, it is still far from a complete product. The predictions are not precise, even on simpler music scores. The reason for this is almost always the seq2seq prediction. Maybe I could have tried to further process the images in some way that could make the predictions better. Either way, both the music score segmentation and the MIDI assembly blocks are independent enough that it should be very easy to swap the seq2seq for a different deep learning OMR model that shares the same input and output formats, so this project can be expanded in the future by trying it on other models.

References

[1] Jorge Calvo-Zaragoza, Jan Hajič Jr., & Alexander Pacha. (2020, julio). Understanding Optical Music Recognition. https://doi.org/10.1145/3397499

[2] Bellini, P., Bruno, I. & Nesi, P. (2001, diciembre). Optical music sheet segmentation. Proceedings First International Conference on WEB Delivering of Music. WEDELMUSIC 2001. https://doi.org/10.1109/wdm.2001.990175

[3] A. Baró, C. Badal and A. Fornés (2020). Handwritten Historical Music Recognition by Sequence-to-Sequence with Attention Mechanism. http://158.109.8.34/people/afornes/publi/conferences/2020_ICFHR_ABaro.pdf

[4] P. Torras, A. Baró, L. Kang, A. Fornés (2021). On the Integration of Language Models into Sequence To Sequence Architectures for Handwritten Music Recognition. http://158.109.8.34/people/afornes/publi/conferences/2021_ISMIR_PTorras.pdf

[5] T. Santaella (2021, july). TFG. Reconocimiento de partituras musicales.

[6] Shatri, E., & Fazekas, G. (2020). Optical Music Recognition: State of the Art and Major Challenges. arXiv. https://doi.org/10.48550/ARXIV. 2006.07885

[7] T. MIKOLOV (2012). PhD. Statistical Language Models Based on Neural Networks. https://www.fit.vut.cz/study/phd-thesis-file/283/283.pdf

[8] Vinaya V et al Int. (2014, May). Journal of Engineering Research and Applications. https://www.ijera.com/papers/Vol4_issue5/Version%203/C045031116.pdf

[9] javidx9. (2016, Feb). https://www.youtube.com/@javidx9

[10] OneLoneCoder/javidx9. (2022, Sep). https://github.com/OneLoneCoder/Javidx9/blob/master/PixelGameEngine/SmallerProjects/OneLoneCoder_PGE_MIDI.cpp

[11] 3Blue1Brown. (2022, 18 noviembre). But what is a convolution? [Vídeo]. YouTube. https://www.youtube.com/watch?v=KuXjwB4LzSA

[12] Phi, M. (2020, 28 junio). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Medium. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[13] javidx9. (2020, Mar) Programming MIDI. [Vídeo]. Youtube. https://www.youtube.com/watch?v=040BKtnDdg0&t=1087s

[14] Official MIDI Specifications. https://www.midi.org/specifications

Bibliography

Understanding LSTM Networks -- colah's blog. (2015). Colah's Blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Phi, M. (2020, 28 junio). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Medium. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Moses, K. (2022, 7 enero). Encoder-Decoder Seq2Seq Models, Clearly Explained!! Medium. https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b

Kostadinov, S. (2021, 7 diciembre). Understanding Encoder-Decoder Sequence to Sequence Model. Medium. https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

Jangid, A. (2021, 11 diciembre). Intuitive Understanding of Seq2seq model & Attention Mechanism in Deep Learning. Medium. https://medium.com/analytics-vidhya/intuitive-understanding-of-seq2seq-model-attention-mechanism-in-deep-learning-1c1c24aace1e

Antonio Ríos Vila, David Rizo, & Jorge Calvo-Zaragoza. (2021, septiembre). Complete Optical Music Recognition via Agnostic Transcription and Machine Translation. https://doi.org/10.1007/978-3-030-86334-0_43

Dirac, L. [Seattle Applied Deep Learning]. (2019, 3 diciembre). LSTM is dead. Long Live Transformers! [Vídeo]. YouTube. https://www.youtube.com/watch?v=S27pHKBEp30