# Integration of Language Models in Sequence to Sequence Optical Music Recognition Systems

Pau Torras Coloma

5 de Febrer de 2021

**Resum–** El present projecte és un estudi del potencial d'integrar per mitjà de diverses tècniques un model de llenguatge a un sistema de Reconeixement Òptic de Partitures (OMR) basat en una arquitectura Sequence to Sequence. L'objectiu és millorar el rendiment del model sobre partitures manuscrites antigues, que són especialment complexes d'interpretar a causa del seu elevat grau de variabilitat i les distorsions que solen incorporar.

**Paraules clau–** Deep Learning, Reconeixement d'escriptura a mà, Reconeixement Òptic de Partitures, Sequence to Sequence, Model de llenguatge, Visió per Computador, Xarxes Neuronals Recurrents

**Abstract–** The following project is a study of the potential of integrating a language model into a Sequence to Sequence-based Optical Music Recognition (OMR) system through various techniques. The goal is to improve the performance of the model on handwritten old music scores, whose interpretation is particularly error-prone due to their high degree of variability and distortion.

**Keywords–** Computer Vision, Deep Learning, Handwriting Recognition, Language Model, Optical Music Recognition, Recurrent Neural Networks, Sequence to Sequence

## 1 INTRODUCTION

OPTICAL Music Recognition (OMR) [1] is a discipline within the field of Document Analysis which consists on the application of Computer Vision techniques to gather and interpret the information stored in images of music score sheets. Although interest in these documents arose as early as the 1960s [2], the field has seen mostly limited attention from the research community due to the fact that music notation is more complex to interpret than regular text writing. The key difference between both systems is that music notation is position-sensitive, in the sense that the meaning of each symbol is affected by its placement in the score.

This has lead to the current situation, where there are many exciting open lines of research and original ideas to be explored and developed. OMR has also been greatly enriched by the recent rise of Deep Neural Network technology, which has provided researchers with better performing models [3], [4], [5].

This project focuses on the application of a Deep-Learning-based OMR [3] system on handwritten old scores, which are particularly difficult to process due to paper degradation, ink stains, corrections, irregular symbol shapes, non-parallel staff lines and perspective distortion, among others. These artefacts are the main cause of a substantial difference in state-of-the-art performance between old and typeset scores. Figure 1 shows a page from J.S. Bach's original *The art of the fugue* and a modern transcription by the OpenScores project in order to better illustrate their differences.

The architecture that powers this work is the Sequence to Sequence (Seq2Seq) [6] model, which has given very promising results on close relatives of OMR, namely Optical Character Recognition (OCR) and Speech Recognition. In these cases, experiments have been conducted where Language Models (LM) [7] , [8] assisted Seq2Seq models

• E-mail de contacte: pau.torras@e-campus.uab.cat
• Menció realitzada: Computació
• Treball tutoritzat per: Alicia Fornés, Arnau Baró (Departament de Ciències de la Computació)
• Curs 2020/21

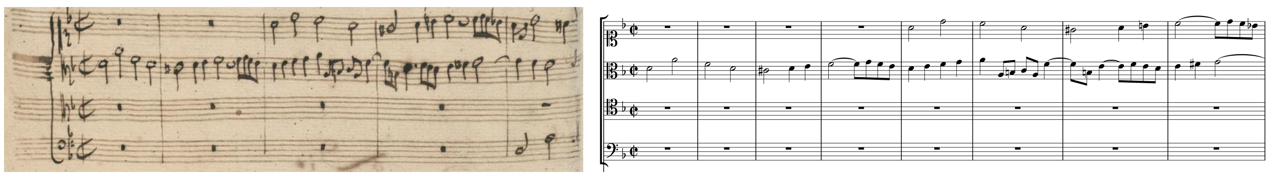


Fig. 1: Example of an Old Handwritten score and its modern transcription

to improve on the final results, but this step has been for the most part omitted for OMR. The integration of said LM in the context of music scores is the key contribution of this work, with the purpose of alleviating possible recognition mistakes and ultimately improving the performance of old handwritten scores recognition.

The structure for this document is the following: section 2 is a description of the objectives set for this project. Section 3 is the breakdown of the methodology that was followed during the project and the planning that was established at the beginning. Section 4 is a summary of the state of the art of OMR prior to this project. Section 5 is an overview on the models which were studied in this work. Section 6 is an overview of the datasets that were used to train the various models. Section 7 is a detailed explanation of the experiments that were performed in order to assess the performance of the models. Section 8 is an explanation of the results that were obtained. Finally, 9 summarises the results and section 10 gives an insight on possible future work, as well as some closing words.

## 2 Objectives

The main goal for this project is to explore the integration of a LM into an OMR system. In order to achieve this, several subsidiary targets were established:

- To perform research on the state of the art for both OMR, Seq2Seq and LM.
- Implementation of various LM integration methods for Seq2Seq models.
- Data preparation for different datasets and training of all integration methods.
- Validation of results gathered from all architectures and comparison with the state of the art.

## 3 Planning and Methodology

This project was developed under an iterative incremental scheme for every integration method. The idea was adding features incrementally while reflecting and improving on the implementation by drawing parallels on previous dev cycles. The original task definition is detailed below:

- **Iteration 1:** For the first iteration, the main tasks were acquainting with data and performing the pre-training step for the LM and the Seq2Seq. This was completely agnostic to the chosen integration method and a requirement to train further models. In parallel, thorough State of the Art research was to be performed and the first integration method developed (Shallow Fusion).

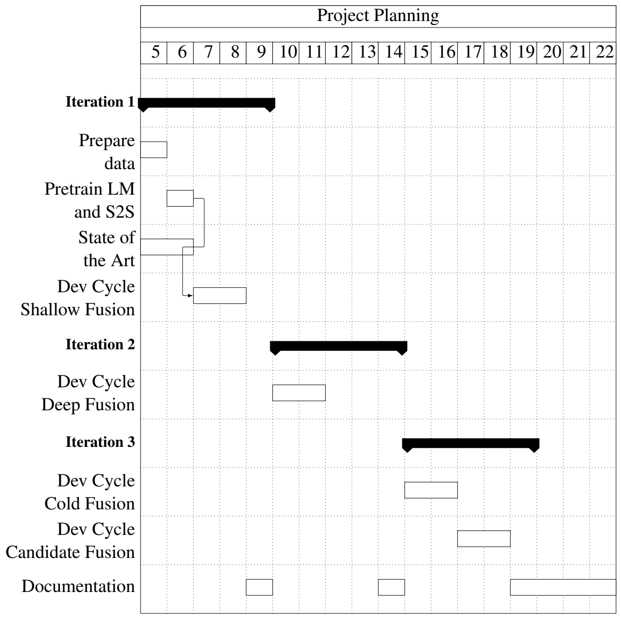


Fig. 2: Gantt Chart for global project planning (weekly). Note that dependencies between data preparation, state of the art and all dev cycles have been omitted for readability, as well as dependencies between dev cycles and full evaluation.

- **Iteration 2:** For the second progress report, Deep fusion was planned for development. Nonetheless, this step suffered considerable delays due to problems in training, which caused some rescheduling on the next milestone.
- **Iteration 3:** For this milestone, Candidate and Cold fusion were planned for development, as well as a joint analysis of model performance. The latter method was finally scrapped in favour of dedicating more time to studying and comparing the others.
- **Preparation of the final exposition and the Poster**.

Figure 2 is a Gantt diagram with temporal details and dependencies among tasks.

In terms of methodology, the project was developed using preexisting code from [3] and implementing the required changes to accommodate the various LM integration systems. The core model is written in Python 3.x [1] using PyTorch [2] for Neural Networks and other Data Science libraries for processing (numpy [3], OpenCV [4], Matplotlib [5]). Other supporting tools were used for the sake of data safety

1 https://www.python.org/
2 https://pytorch.org/
3 https://numpy.org/
4 https://opencv.org/
5 https://matplotlib.org/

and workflow: Git for code version control and PyCharm / Jupyter Notebook for code writing. Training was partially performed on the Computer Vision Center compute cluster

Each development cycle consisted on checking a method's theoretical background, writing the corresponding code, verifying its functionality with a few epochs of training and launching the full training task once verified.

## 4 State of the Art

Prior to the Deep Learning "revolution" of the early 2010s there was mainly one common architecture for OMR, which consisted on a set of well-established steps. In their 2012 review of OMR paper, Rebelo et al. [1] summarised them as image preprocessing, music symbol recognition, music notation reconstruction and final representation construction.

This structure was challenged with the advances made in Deep Learning, which allowed researchers improve both on each phase of the pipeline individually (for instance, providing pixel-level binarization of music scores through Convolutional Neural Networks (CNN) [9] or providing a system to associate segmented symbols into a graph for notation reconstruction [10]) and also through architectures capable of performing the entire pipeline in a single step. Some examples can be found in the work of Van der Wel *et al.* [4], who used a (CNN) stacked with a Seq2Seq model in order to perform OMR on typeset scores, and Huang *et al.* [11], who presented a single-step CNN-based OMR architecture which obtained state-of-the-art results on monophonic typeset scores.

Fewer have ventured in the realm of OMR for handwritten scores using end-to-end deep neural models. Most notable works include the effort by Calvo-Zaragoza *et al.* to recognise old scores in mensural notation using a CTC-based model backed by an $n$-gram language model, and the work by Baró *et al.* [3] expanding on CTC and also introducing an Attention-based [12] Seq2Seq architecture.

In terms of Language Modelling, the earliest instance of said technique in the context of a recognition task is [13], which used n-grams in order to make OCR machines context-aware and therefore more robust to recognition mistakes. Since n-grams are fairly cheap to implement and give reasonably good results, they have been used fairly consistently even in recent times [5], although with the rise of DNN technology other approaches based on neural language models have been tested. Examples which use RNNs in conjunction with various recognition techniques include [8] and [14].

## 5 Models

The following section is devoted to describing the models studied during the project.

### 5.1 Recurrent Neural Networks

Recurrent Neural Networks (RNN) [12] are a neural network architecture family characterised by the existence of temporal dependence between predictions. In other words, a prediction may be influenced both by the input at the current time step (if any) and the previous state of the model. This architecture is particularly useful for many reasons: it allows neural networks to have an arbitrary number of input and output elements (because they can be segmented into single vectors that are fed or outputted in consecutive time steps) and it provides a natural way of processing of data with strong sequential interdependence.

At the same time, this architecture has some drawbacks. Since elements in a sequence need to be fed in consecutive time steps, they may take longer to train than other architectures that can process the entire input sequence at once. Moreover, in their simpler expression, RNNs have a limited capacity to save information from very distant sequence elements (as well as suffering from vanishing gradient problems).

To alleviate the aforementioned effects, more sophisticated models have been devised with idea of strengthening the system's capacity to control the information that gets sent into further iterations. In this project, Gated Recurrent Units (GRU) were used [12], which provide a series of gate mechanisms in order to control the amount of information from the previous state that is no longer relevant and needs to be deleted (reset gate) and the amount of information from the previous time step that needs to be considered in the current time step (update gate). This adaptation capacity allows the model to keep important information through longer time intervals. Also, since new data is generated through a Hyperbolic Tangent activation function, its range is forced into the interval $[-1, 1]$ and the problem of exploding/vanishing gradient is much less significant.

### 5.2 Theoretical Background: Sequence to Sequence

Seq2Seq [6] models are an Encoder-Decoder architecture, which is a family of models that is characterised by being split into two modules in which one generates an intermediate representation of the input data ("Encoder") and the other generates the output in accordance to the intermediate representation ("Decoder"). As its name implies, Seq2Seq architectures are used for situations where an arbitrary length output sequence can be obtained from the transformation of an arbitrary length input sequence. This model was originally conceived for Natural Language Processing (neural machine translation, speech recognition, etc.) but has seen some success outside the field.

The Encoder is a bidirectional stack of recurrent units, which processes the input sequence both "left-to-right" and "right-to-left" so as to have dependence information from all time steps in the intermediate representation. In the original concept of Seq2Seq, said representation was usually a single vector of data which was updated with each iteration of the Encoder. Once the input sequence is completely processed, the Decoder module, which is usually a unidirectional stack of recurrent units, receives a special token ("Go") as input alongside the intermediate representation, from which it generates the first token of the output sequence. It then generates tokens from its previous output and the intermediate representation until a special "End" token is predicted.

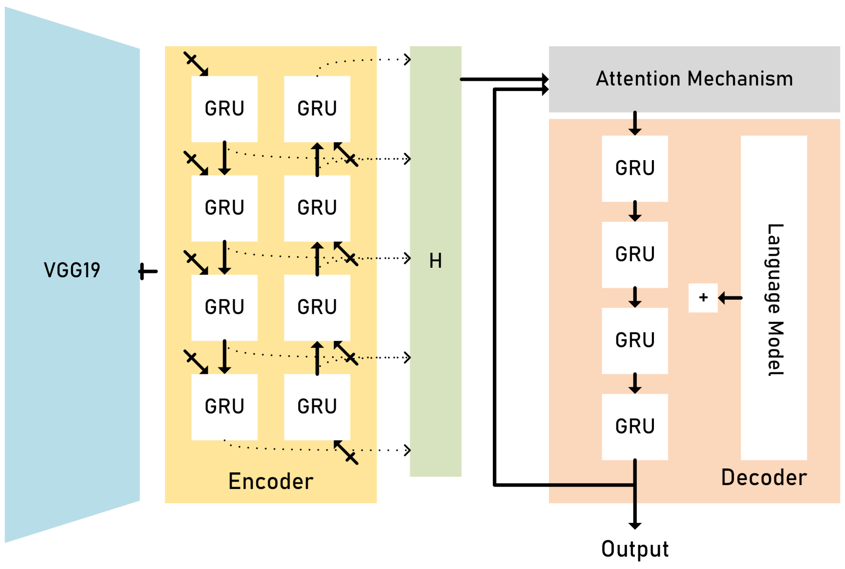


Fig. 3: Sequence to Sequence model summary

## 5.3 Attention Mechanism

As mentioned earlier, in its original conception, Seq2Seq stored information in a single limited vector that attempted to compress the whole input sequence and then perform inference conditioned by it. Attention mechanisms [15] are a way of sidestepping this limitation by letting the model decide what information is most relevant at inference time (most specifically, the Decoding step).

Instead of encoding the information in a single vector, each encoding step generates a feature vector that gets appended to the hidden state of the Seq2Seq. Then, an attention system can weight each of the feature vectors in order to consider only the subset of required information for every inference step.

## 5.4 Sequence to Sequence applied to Music Scores

Seq2Seq models that have been exposed thus far do not consider the input to be an image, but rather a generic collection of input vectors. In [3], the publication that serves as the backbone for this work, a system that incorporates all of the aforementioned techniques is crafted which also provides support for images as inputs and is tuned for OMR. Figure 3 is a depiction of the model alongside this work's contribution of Language modelling and integration.

The model performs the following steps:

- A data augmentation system generates a unique variation of an input image by applying noise and geometric transformations (only during training).
- The altered score image is fed into a Convolutional Neural Network, which is based on a VGG19 [16] albeit with the last max pooling layer removed. This Convolutional network outputs a set of feature vectors that will serve as the input sequence for the recurrent segment of the model.
- The convolutional feature vectors are fed into the bidirectional Encoder. This segment generates an intermediate representation of the input sequence as a set of vectors which forms the state of the model.
- When the Encoder is finished processing the input, the state is fed iterativelly alongside the last predicted token in the sequence into the Decoder, whose output is the final prediction. In order to assess which of the input vectors are more relevant for each time step, an attention mechanism weights the state vectors. This is performed using location attention by Chorowsky et al [17], which uses the previous attention vector, the Decoder's current hidden state and its previous output.
- During the Decoding step, a LM may be implemented and integrated using one of the integration methods that shall be discussed below.

## 5.5 Language Model

LMs are systems that provide the probability of a token given a sequence of previous tokens. Let $M$ be a LM and $y_t$ the predicted token at time step $t$, then

$$M(y_1 \dots y_{t-1}) \approx P(y_t | y_1 \dots y_{t-1}). \qquad (1)$$

Many techniques exist for language modelling, but since RNNs have proven to be an effective model, this work focuses only on a single architecture based on four stacked GRUs.

## 5.6 Language Model Integration

The task of Language Model integration with Seq2Seq models with the purpose of improving the former's performance has been explored with various approaches. The following section is devoted to listing those that were found most interesting and were considered for implementation. Figure 4 shows a graphical representation of all methods.

- **Shallow Fusion** (Gulcehre et al. [7]): This is the simplest and most straightforward method of integration. Both models (LM and Seq2Seq) are kept independent and the final prediction is calculated as

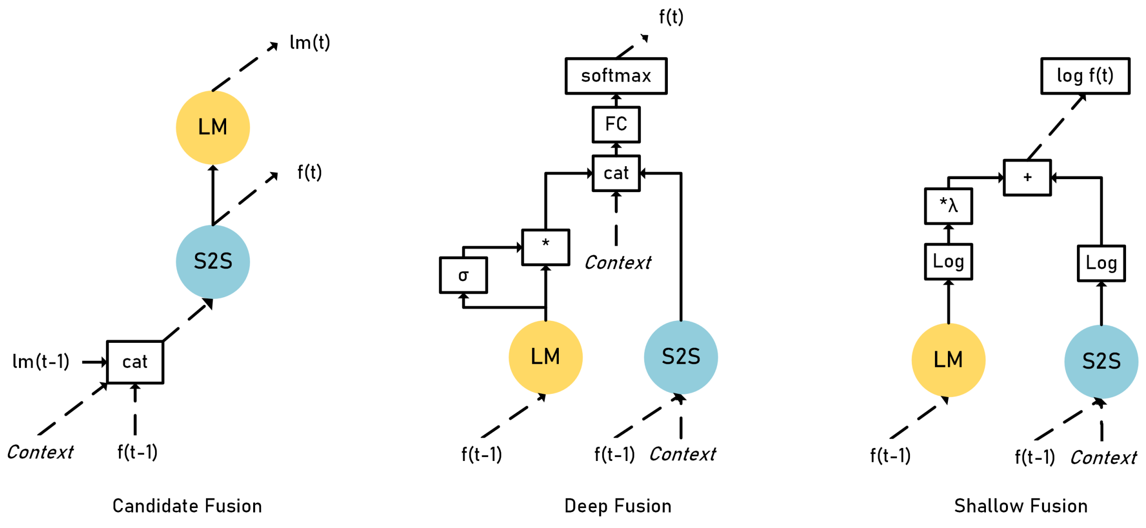


Fig. 4: Language model integration summary

$$\log P\left(y_t|y_1...y_{t-1}\right) = \log P_{CL}\left(y_t|y_1...y_{t-1}\right) + \lambda \log P_{LM}\left(y_t|y_1...y_{t-1}\right) \tag{2}$$

where $P_{CL}$, $P_{LM}$ are the outputs of the classifier and the language model respectively (interpreted as probabilities). This method of integration requires pre-training for both modules, and then the $\lambda$ parameter is fine-tuned.

- **Deep Fusion** (Gulcehre et al. [7]): This is an extension of the Shallow Fusion approach that attempts to further merge both models. It uses the hidden states of both models for the final prediction. Let $\sigma$ be the sigmoid activation function, sm be the softmax function and $W_{DF}$ and $b_{DF}$ be learnable parameters

$$P(y_t|y_1...y_{t-1}) = \mathrm{sm}(W_{DF}h_t^{DF} + b_{DF}). \tag{3}$$

The Deep Fusion hidden state $h_t^{DF}$ is the result of concatenating the Sequence to Sequence (Seq2Seq) context vector $c_t$, the Classifier's hidden state $h_t^{CL}$ and a gated version of the Language Model (LM)'s hidden state, as seen in

$$h_t^{DF} = \left[c_t; h_t^{CL}; g_t h_t^{LM}\right]. \tag{4}$$

The LM gate $g_t$ is in its turn computed as

$$g_t = \sigma(v_g^T h_t^{LM} + b_g) \tag{5}$$

where $v_g$ and $b_g$ are learnable parameter vectors.

This method also requires both the LM and the classifier to be properly pre-trained.

- **Candidate Fusion** (Kang et al. [18]). This approach differs greatly from the ones preceding it, in the sense that previous methods focused on merging the outputs of both models, while this one reinforces the task of the Decoder by feeding it the probabilities obtained by the LM as

$$\hat{h}_t = \mathrm{Decoder}(\left[c_t, y_{t-1}, p_{t-1}^{lm}\right], h_{t-1}) \tag{6}$$

where $c_t$ is the current context vector, $y_{t-1}$ is the previous prediction and $p_{t-1}^{lm}$ is the probability distribution obtained by the LM with the output of the previous time step.

This method also requires both the LM and the classifier to be properly pre-trained.

## 6 Data

The datasets that were used to train the various models and assess their accuracy are the following (See figure 5):

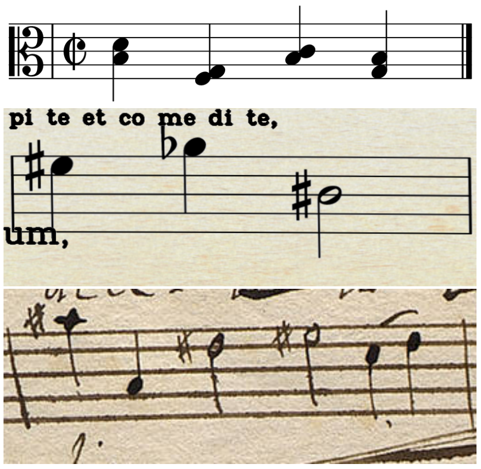
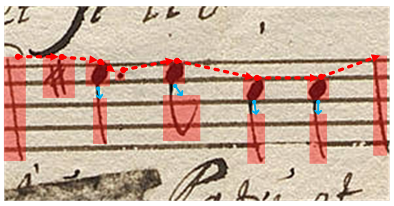


Fig. 5: Sample measures from the SM, SO and HW datasets respectively

- **Synthetic Modern (SM):** A dataset consisting of typeset scores. They are synthetically-generated polyphonic measures that contain a very wide array of symbols including clefs, accidentals, ornament notes and rests.
- **Synthetic Old (SO):** A dataset consisting of typeset scores similar to those found in the SM dataset, but distorted with typical paper and ink degradation artefacts to make them look like old handwritten sheet music.



```
barline_light.noNote, epsilon, sharp.S4, epsilon,
noteheadBlack.S4, steamQuarterHalfDown.noNote, epsilon,
dot.noNote, epsilon, noteheadBlack.S4, flag8thDown.noNote,
epsilon, noteheadBlack.S3, steamQuarterHalfDown.noNote,
epsilon, noteheadBlack.S3, steamQuarterHalfDown.noNote,
epsilon, barline_light.noNote
```

Fig. 6: Example of the organisation of tokens in the ground truth file. Red areas limit the span of a primitive. Red arrows indicate epsilon steps and blue arrows indicate primitives which are stored sequentially between epsilons

- **Handwritten (HW):** A collection of measures from real handwritten scores from Pau Llinàs, the Kapellmeister of Santa María del Pi in Barcelona during the first half of the 18th century. These measures come from a liturgical choral work, which implies it is monophonic, it contains "voice" clefs and it contains musical motifs within its time and context musical trends.

- **Adjusted Synthetic Modern (ASM):** A subset of the SM dataset which was built on samples closer to those found in Pau Llinàs' scores. More detail is provided in section 7.

Ground truth data is provided as a flat context-free sequence of tokens arranged in top-to-bottom left-to-right order. The rationale behind this system is exemplified in Figure 6. The idea is the special epsilon token tells the classifier the next set of tokens is located on a different column. Another special token exists to account for single-staff multiple-voice works, which is the vertical epsilon.

The symbols these datasets provide are the basic musical primitives that form more elaborate symbols. For instance, notes are split into noteheads (as seen in Figure 6, its pitch is indicated by appending the Space or Line in which it is placed after a dot), flags (its orientation, upwards or downwards is indicated as well), stems and, if notes are tied, beams and their direction and value. There are various single-token symbols, which are barlines, accidentals, dots, rests, time signatures and other less frequent elements. For compound symbols like slurs and beams, an opening and closing token is required before and after the group of elements that conform the symbol.

## 7 Experiments

To assess the performance of the various integration methods six experiments were performed under two training strategies, which differ on the dataset that was used to train the LM (SM or ASM respectively). For Shallow Fusion, both experiments were performed with a $\lambda = 0.1$.

### 7.1 Experiments on the SM dataset

Since the target of the project is to improve results on handwritten scores, a training strategy was devised to prevent optimisation toward synthetic scores. All integration methods require a pretrained LM and recogniser, which was performed under the SM dataset. Then, a joint training phase was deployed which used both the SO and the HW datasets in parallel. The idea was to try to use the SO dataset as the backbone for training, as it has more samples, while making the most out of the few HW samples available.

Initially, the models were trained using a pool with 90% of the samples belonging to the SO dataset and the remaining 10% to the HW dataset. Every 10 epochs the proportion of SO scores decreased by 10% over the total, down to 10%. Since the number of SO samples is substantially higher than the number of HW samples, the latter were duplicated randomly whenever not enough of them were available. The incorporated image augmentation system for training helped prevent overfitting.

In order to keep track of the quality of the models and pick the best performing ones, validation and test were performed on HW dataset samples only.

Note also that the reason for not using datasets comprised of samples coming from only one dataset was because previous experiments showed substantial performance hits.

### 7.2 Experiments on the ASM dataset

This batch of experiments was performed under the same training strategy seen in 7.1, but a new dataset was compiled to pretrain the LM. The idea was to try to find the samples from the SM dataset which were more closely aligned to the HW dataset. This was done because the HW dataset is crafted from scores written in the 18th century for choral interpretation in a religious context, while the SM dataset is an arbitrary set of scores with differing characteristics.

In order to select those samples which were most relevant, a selection study was performed in which the statistical distribution of tokens in the original HW dataset was calculated. Then, the 66% of measures the SM dataset which contained the highest ratio of tokens present in the HW dataset were selected. This method has some drawbacks, in the sense that it does not completely eliminate the presence of foreign tokens. Nevertheless, it should be noted that the highest interest for the LM is to be provided with the biggest number of correct sub-sequences possible. By establishing a more aggressive pruning strategy many relevant sequences may have been lost and therefore the LM may have underperformed.

## 8 Results

Table 1 shows the performance in Symbol Error Rate (SER) of the best epoch in each training iteration. SER is defined as

$$SER(\%) = \frac{I + R + S}{T} \cdot 100 \quad (7)$$

where $I$, $R$ and $S$ are the minimal number of token insertions, removals and substitutions required to transform

TABLE 1: SUMMARY OF PERFORMED EXPERIMENTS AND RESULTS IN SER(%). LOWER IS BETTER.

| Method | Dataset | 90-10 | 80-20 | 70-30 | 60-40 | 50-50 | 40-60 | 30-70 | 20-80 | 10-90 | 0-100 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Baseline [3]** | - | 60.03 | - | - | 66.20 | - | 43.38 | - | 37.86 | 34.56 | **31.79** |
| **Deep** | SM | 31.30 | 28.52 | 29.87 | 29.37 | 28.05 | **26.11** | 27.74 | 27.37 | 28.32 | - |
| **Shallow** | SM | 36.79 | 32.91 | 33.27 | 33.36 | 31.76 | 32.75 | 30.87 | 30.72 | **30.58** | - |
| **Candidate** | SM | 33.50 | 28.93 | 28.64 | 28.08 | 27.48 | 26.82 | 27.23 | 26.61 | **25.80** | - |
| **Deep** | ASM | 28.24 | 29.53 | 27.82 | 27.36 | 25.95 | 27.21 | 25.63 | 25.15 | **25.54** | - |
| **Shallow** | ASM | 35.34 | 34.75 | 36.67 | **32.42** | 34.23 | 34.52 | 33.76 | 33.79 | 35.13 | - |
| **Candidate** | ASM | 32.07 | 28.61 | 28.71 | 27.55 | 27.71 | 27.20 | 27.77 | 28.04 | **25.73** | - |

the predicted sequence to the ground truth sequence. These results are analysed in subsection 8.1.

A qualitative analysis of the results was performed to try to study the reason for the differences in performance among the various methods. Details are shown in 8.2

A perplexity study was also performed in order to assess the difficulty of the LM task *a priori*. In order to try to evaluate the level of confidence on the predictions the various models have, a different study based on the perplexity of the classifier and the LM was performed. Results for this analysis are shown on section 8.3

## 8.1 Quantitative Analysis

The baseline result from [3] is shown on the first row of the Table, which yields a best of 31.79% of SER. Most experiments managed to obtain better results, with Deep and Candidate fusion being the better performing ones.

The general pattern is that earlier iterations perform worse than latter ones. There are a few exceptions, which are the SM version of Deep Fusion and the ASM version of Shallow Fusion, which have better results in intermediate phases. This might be caused by local minima, which the model optimises out of in the following epochs.

Another general remark is that models pretrained with the ASM dataset seem to perform slightly better (except for Shallow Fusion), although this difference could be also attributed to optimisation, since it is neither substantial nor absolute.

The most remarkable result is the 6% decrease in SER that has been obtained in both Deep and Candidate fusion for both pretraining datasets, but further study needs to be performed in order to assess its causes.

## 8.2 Qualitative Analysis

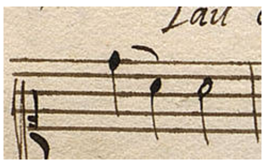
In order to help illustrate the advantages and the flaws of the various integration mechanisms, an example shall be presented through a full transcript of a measure using the ASM version of Deep Fusion and Shallow Fusion.

Figure 7 shows the transcription of the image done by the ASM-trained Deep and Shallow integration methods. This bar is particularly interesting because it has various types of notes and it has a slur, which is a particular type of token whose start is classified separately from its end, requiring some context awareness on top of correct recognition.

In general, the transcription by the Deep classifier can be assessed as fairly accurate: the only mistakes are the last notehead, which is assumed to be a quarter note two spaces above the real one, and the last stem, which is interpreted as the final bar line and the closing of the bar. The LM provided a remarkably precise evaluation of the line as well, with only an incorrect bar line at the second position and incorrect positions of the notes.

Shallow fusion's version omits the opening and closing slurs, and also seems to miss the correct placement of notes. Overall, the model seems to follow the LM strictly when appropriate, and the classifier performs in a similar manner as the Deep Fusion version. The same barline mistake is also found in line 3 of the LM output, which is unsettling but may be attributed to the fact that the model may not expect its original barline output to be overridden by a clef and therefore outputs a high-probability token.

Generally speaking, the model can understand the semantic relationship between tokens (for instance, after a sequence of primitives that forms a symbol an epsilon will be added). Moreover, most mistakes seem related to recognition. This is especially true with noteheads, which are



| | Deep Fusion | | | | Shallow Fusion | | | |
|---|---|---|---|---|---|---|---|---|
| **Ground Truth** | **Classifier** | $P_c$ | **LM** | $P_{LM}$ | **Classifier** | $P_c$ | **LM** | $P_{LM}$ |
| C-Clef.L1 | C-Clef.L1 | 0.599 | barline_light | 0.850 | C-Clef.L1 | 0.601 | barline_light | 0.864 |
| epsilon | epsilon | 0.599 | epsilon | 1.000 | epsilon | 0.592 | epsilon | 1.000 |
| startSlur | startSlur | 0.024 | barline_light | 0.164 | noteheadBlack.L4 | 0.337 | barline_light | 0.170 |
| epsilon | epsilon | 0.452 | epsilon | 1.000 | steamQuarterHalfDown | 0.588 | steamQuarterHalfDown | 0.986 |
| noteheadBlack.L5 | noteheadBlack.L5 | 0.054 | noteheadBlack.L4 | 0.997 | epsilon | 0.595 | epsilon | 1.000 |
| steamQuarterHalfDown | steamQuarterHalfDown | 0.443 | steamQuarterHalfDown | 0.997 | noteheadBlack.L4 | 0.496 | noteheadBlack.L4 | 0.984 |
| epsilon | epsilon | 0.601 | epsilon | 1.000 | steamQuarterHalfDown | 0.583 | steamQuarterHalfDown | 0.529 |
| noteheadBlack.S3 | noteheadBlack.S3 | 0.417 | noteheadBlack.S4 | 0.499 | epsilon | 0.599 | epsilon | 1.000 |
| steamQuarterHalfDown | steamQuarterHalfDown | 0.615 | steamQuarterHalfDown | 1.000 | noteheadBlack.S3 | 0.132 | halfRest | 0.972 |
| epsilon | epsilon | 0.625 | epsilon | 1.000 | steamQuarterHalfDown | 0.580 | steamQuarterHalfDown | 0.994 |
| endSlur | endSlur | 0.478 | endSlur | 1.000 | epsilon | 0.599 | epsilon | 1.000 |
| epsilon | epsilon | 0.430 | epsilon | 1.000 | barline_light | 0.481 | barline_light | 0.271 |
| noteheadHalf.S3 | noteheadBlack.S5 | 0.040 | noteheadBlack.S5 | 0.706 | | | | |
| steamQuarterHalfDown | barline_light | 0.181 | steamQuarterHalfDown | 1.000 | | | | |
| | **SER**: | 18.750% | | | **SER**: | 43.750% | | |

Fig. 7: Measure from the HW dataset with its transcriptions and probabilities (Deep + ASM and Shallow + ASM models). Some of the relevant mistakes are highlighted in red and orange depending on the token where the mistake is found

commonly misplaced a few steps above or below their real position. Other cases involve less frequent tokens which the recogniser might not be properly acquainted with (such as sharps, which in some cases are interpreted as whole notes).

Noteheads are also difficult to properly classify by their duration. The model does not seem to understand the concept of beat, since it outputs notes that would last longer than the bar indicates. The LM seems equally affected by this phenomenon.

In general, the hardest task is determining the content after each epsilon. Drawing a parallel to OCR, the epsilon character acts like a space, after which many words may fit the current context. Whenever a mistake is made after an epsilon, a significant amount of error may be accumulated because the model strays from the original context (in the example, the fact that a stem was classified as a barline altered the context of the sentence and forced an end token). The LM cannot help in the situation mentioned in the former issue, but it can help keep track of certain contextual tokens (slurs or the opening and closing beams for compound notes).

It can also be observed that a high confidence by the LM does not imply that the model will weight its output as correct, which is expected since integration methods are meant to balance the predictions of both models.

## 8.3 Perplexity Analysis

Since there is a large proportion of issues that are closely related to the probability distribution of tokens, I decided to investigate a little bit further by analysing the perplexity of the test dataset.

Perplexity is a measure of the degree of "choice" a model faces when performing a transcription task. It can be interpreted as the mean number of options from which to choose at a certain time step. Let $s$ be a text sequence of $N$ words (in this case, the whole test dataset concatenated) and $P(x_1 \dots x_N)$ the joint probability of the text sequence calculated from the training dataset, the perplexity of the sequence $PP(s)$ is defined as

$$PP(s) = \sqrt[N]{\frac{1}{P(x_1, \dots x_N)}}. \tag{8}$$

Since the joint probability is difficult to calculate, a second degree Markov assumption is made, so that every text token depends only of its direct predecessor. This assumption was made because in the HW dataset, the maximum number of tokens between epsilons is 2, which implies that only one relevant token of information may precede a token. Through this assumption and the chain rule, $P(x_1 \dots x_N)$ can be calculated as a bigram, expressed as

$$P(x_1 \dots x_N) \approx P(x_1) \cdot P(x_2|x_1) \cdot \dots \cdot P(x_N|x_{N-1}). \tag{9}$$

This was done considering training with the HW dataset and the whole mix of HW, SM and SO datasets. The results were a perplexity of $4.05$ training with HW data and $6.02$ training with the joint dataset.

The meaning of a $4.05$ perplexity when training with HW data is that, by training with this dataset, the average amount of options is 4. Upon a closer inspection of the various bigram distributions, an interesting conclusion can be drawn (which was mentioned earlier as a part of the qualitative analysis): the deviation in the level of choice is extremely high depending on the bigram. There are certain probabilities close to 1, such as the probability of an epsilon after any atomic token, but there are many tokens that can be succeeded by a significant amount of options. These usually revolve around the various versions of noteheads, which the LM cannot properly analyse. These have also been the tokens that the classifier has struggled the most with.

When using a broader dataset, the level of perplexity increases, in this case from $4.05$ to $6.72$, because the model has more information about the possible combinations of bigrams and therefore has a broader choice. This was part of the motivation behind creating a "closer" dataset.

In order to try to assess the confidence with which a model predicts every token, a different take on perplexity was used, which could be then compared to the theoretical value obtained with the bigram model. Let $d$ be the number of unique tokens in the dataset, $z$ the hidden size and $M : h, s \to \mathbb{R}^d$ where $h \in \mathbb{R}^z, s \in \mathbb{R}^d$ be the model in question. If one considers $M(h_t, s_{t-1}) \approx P(X|x_1...x_{t-1})$, then

$$P(seq) \approx \prod_{i=1}^{t} C(h_t, s_{t-1}) \tag{10}$$

which can be used with equation 8 to calculate the empirical perplexity of the model. This was done for the LM and also for the classifier as a whole, even if in this context the aforementioned assumptions do not hold, for illustrative purposes. Results are shown in Table 2.

The main observation from this Table is that the LM has a similar level of perplexity as the theoretical threshold that was calculated. This implies that the training process optimises the LM towards HW scores correctly even when it uses samples from all datasets. The classifier, on the other hand, suffers from having a much wider array of options in most cases. Even more remarkable is the fact that higher perplexity does not imply lower classifier performance, since Candidate Fusion is actually one of the best performing methods.

## 9 Discussion

After analysing the results from all experiments, some general conclusions can be drawn. The most relevant result

TABLE 2: Empirical perplexity for the LM and the classifier

| | SM | | | ASM | | |
|---|---|---|---|---|---|---|
| | Deep | Shallow | Candidate | Deep | Shallow | Candidate |
| LM | 4.21 | 4.58 | 4.26 | 4.38 | 4.13 | 3.85 |
| Classifier | 5.79 | 5.06 | 7.63 | 4.89 | 5.72 | 7.80 |

is the 6 SER(%) improvement obtained by adding a LM through Candidate or Deep fusion. Nevertheless, it is hard to assess the reason why some LM integrations perform better than others outside of a very subjective qualitative study. A perplexity analysis showed that the model which had a higher degree of choice was also one of the best performing, which is contrary to what would be expected. Shallow fusion seems to underperform because it does not take the LM sufficiently into account, for which I believe a more thorough hyperparameter search (especially for the $\lambda$ parameter) might be required.

The LM is good at palliating some syntactic mistakes (such as tokens that require a specific successor) or providing information on tokens that are very frequent (which the decoder can learn by default). However, there is a set of possible recognition mistakes where the LM is not helpful. These are mostly related to aesthetic arbitrary aspects of music, such as notes, note lengths, expressiveness, finding accidentals and beat. The latter could perhaps be enforced through other means.

The adjustment strategy on the ASM dataset showed no significant improvement. Instead, training with more handwritten data might help achieve better results. This data should also take into account that the time period matches that of the target scores that will be recognised, as well as its style and context.

## 10 Conclusion

The main objective for this project, which was exploring the possibility of integrating a LM into a Seq2Seq-based handwritten OMR system, has been successfully accomplished. Three different LM integration methods have been implemented and validated, and their final results have actually improved on the state of the art. This proves that a LM can effectively help palliate recognition mistakes on handwritten music scores, although there is still work to be done in order to lower error rates even further. A publication has been submitted to ICDAR 2021 from the content of this project [19].

From the results shown in the Discussion section, I have considered a few future lines of work. Since the beat seems to be difficult to enforce through Deep Learning methods, a combination of the logit output of the classifier and a conventional grammar system with basic semantic analysis could be used to enforce the correct amount of time per measure, as well as some other syntactical mistakes. Moreover, Candidate Fusion has the ability to provide the classifier with future information. Perhaps this could be fed to the attention mechanism to help it "find" possible tokens. Lastly, another possibility would be to drop the Seq2Seq architecture and attempt to perform OMR by using Transformers, which have been obtaining outstanding results in many NLP-related fields.

This project has been my first true contact with Deep Learning research and it has given me the opportunity to combine two of my greatest interests, which are music and AI. I have learnt a lot about many exciting topics within computer vision, natural language processing and data science, which I want to devote my professional life to. I have also learnt that mistakes are bound to happen, for which one should be prepared and watchful.

Overall, I am happy with the obtained results and I expect to carry on this topic of research one way or another during the following months.

## Acknowledgements

I would like to thank my project tutors, Alicia and Arnau, for their immense support and patience during this project, my colleague Lei, for his many insightful comments and the great help he has provided me with and my sister Alba for her help with a proofread of this document.

## References

[1] A. Rebelo, I. Fujinaga, F. Paszkiewicz, A. R. Marcal, C. Guedes, and J. S. Cardoso, "Optical music recognition: state-of-the-art and open issues," *IJMIR*, vol. 1, no. 3, pp. 173–190, 2012.

[2] D. H. Pruslin, G. Read, D. Prerau, W. Hastings, J. Mahoney, B. Widrow, M. E. Hoff, D. Rumelhart, G. Hinton, R. Williams, *et al.*, *Automatic recognition of sheet music*, vol. 21. Taplinger, 1966.

[3] A. Baró, C. Badal, and A. Fornés, "Handwritten historical music recognition by sequence-to-sequence with attention mechanism," in *ICFHR*, pp. 205–210, 2020.

[4] E. van der Wel and K. Ullrich, "Optical music recognition with convolutional sequence-to-sequence models," in *ISMIR*, pp. 731–737, 2017.

[5] J. Calvo-Zaragoza, A. H. Toselli, and E. Vidal, "Handwritten music recognition for mensural notation with convolutional recurrent neural networks," *Pattern Recognition Letters*, vol. 128, pp. 115–121, 2019.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NeurIPS*, pp. 3104–3112, 2014.

[7] C. Gulcehre, O. Firat, K. Xu, K. Cho, L. Barrault, H.-C. Lin, F. Bougares, H. Schwenk, and Y. Bengio, "On using monolingual corpora in neural machine translation," *arXiv preprint arXiv:1503.03535*, 2015.

[8] L. Kang, J. I. Toledo, P. Riba, M. Villegas, A. Fornés, and M. Rusinol, "Convolve, attend and spell: An attention-based sequence-to-sequence model for handwritten word recognition," in *GCPR*, pp. 459–472, Springer, 2018.

[9] J. Calvo-Zaragoza, G. Vigliensoni, and I. Fujinaga, "Pixel-wise binarization of musical documents with convolutional neural networks," in *MVA*, pp. 362–365, 2017.

[10] A. Pacha, J. Calvo-Zaragoza, and J. Hajic Jr, "Learning notation graph construction for full-pipeline optical music recognition.," in *ISMIR*, pp. 75–82, 2019.

[11] Z. Huang, X. Jia, and Y. Guo, "State-of-the-art model for music object recognition with deep learning," *Applied Sciences*, vol. 9, no. 13, p. 2645, 2019.

[12] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *EMNLP*, pp. 1724–1734, 2014.

[13] C. Y. Suen, "n-gram statistics for natural language understanding and text processing," *PAMI*, vol. 1, no. 2, pp. 164–172, 1979.

[14] T. Hori, J. Cho, and S. Watanabe, "End-to-end speech recognition with word-based rnn language models," in *SLT*, pp. 389–396, 2018.

[15] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," *CoRR*, vol. abs/1508.04395, 2015.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *NeurIPS*, vol. 28, pp. 577–585, 2015.

[18] L. Kang, P. Riba, M. Villegas, A. Fornés, and M. Rusiñol, "Candidate fusion: Integrating language modelling into a sequence-to-sequence handwritten word recognition architecture," *Pattern Recognition*, vol. 112, p. 107790, 2021.

[19] P. Torras, A. Baró, L. Kang, and A. Fornés, "On the integration of language models into sequence to sequence architectures for handwritten music recognition," in *ICDAR (submitted)*, 2021.