GAN-based Content-Conditioned Generation of Handwritten Musical Symbols

Gerard Asbert

Computer Vision Center, Universitat Autònoma de Barcelona, Spain

Abstract—Optical Music Recognition (OMR) is a field with a serious problem regarding the lack of real annotated data. We want to address this problem by trying to generate synthetic musical data that is as close in visual resemblance and in training value for deep learning models as real musical scores. Generative Adversarial Networks (GANs) have been successfully applied in various domains, however, OMR still remains an area with limited exploration in the field of data generation. And even though there have been some studies relating GANs with OMR, they are still unable to generate realistic yet diverse images of handwritten musical symbols. In this study, we aim to advance the state of the art on OMR-related GANs by gathering a diverse and high-quality dataset of real handwritten musical symbols, applying data augmentation techniques to address the scarcity of training data, designing and training a GAN architecture tailored to the generation of synthetic handwritten musical symbols, combining these generated symbols into musical staves, and finally, evaluating the visual quality and the research value of these completely synthetic staves.

Keywords—Generative Adversarial Networks, Handwritten musical symbols, Content conditioning, Optical Music Recognition, Music sheet generation, Automated model saving

Resum— L'Optical Music Recognition (OMR) és un camp amb un greu problema pel que fa a la manca de dades anotades reals. Volem abordar aquest problema intentant generar dades musicals sintètiques que siguin tan semblants en aspecte visual i en el valor que poden aportar a l'entrenament de models d'aprenentatge profund com les partitures musicals reals. Les Generative Adversarial Networks (GAN) s'han aplicat amb èxit en diversos dominis, tanmateix, l'OMR continua sent una àrea amb una exploració limitada en l'àmbit de la generació de dades. I tot i que hi ha hagut alguns estudis que relacionen les GANs amb l'OMR, encara no poden generar imatges realistes però diverses de símbols musicals escrits a mà. En aquest estudi, pretenem avançar en l'estat de l'art de les GANs relacionades amb l'OMR recopilant un conjunt de dades divers i d'alta qualitat de símbols musicals reals escrits a mà, aplicant tècniques d'augment de dades per abordar l'escassetat de dades d'entrenament, dissenyant i entrenant una arquitectura GAN adaptada a la generació de símbols musicals manuscrits sintètics, combinant aquests símbols generats en pentagrames musicals i, finalment, avaluant la qualitat visual i el valor de recerca d'aquests pentagrames completament sintètics.

Paraules Clau—Generative Adversarial Networks, Símbols musicals manuscrits, Content conditioning, Optical Music Recognition, Generació de partitures, Automated model saving

1 Introduction

Optical Music Recognition (OMR), as a field, consists on getting an image of a musical score, whether it is printed or handwritten, and automatically recognizing its contents. Recently, it has been getting more attention as, if it manages to keep up with the latest technological advancements, it could revolutionize the way we see and write music. Not only could it help composers to efficiently capture all the intricate ideas that could potentially form a piece of music, but it could do so in a piece of software that lets the user rapidly edit or change their score sheets as they wish.

- *E-mail de contacte: gasbert@cvc.uab.cat*
- *Menció realitzada: Computació*
- *Treball tutoritzat per: Alicia Fornés (Document Analysis Group)*
  Pau Torras (Document Analysis Group)
- *Curs 2024/25*

From a research point of view, the main use that OMR can bring to the table is the transcription of historical handwritten scores. Here in Catalonia, thousands of musical scores ranging from the 9th to the 21st century are scattered throughout different libraries across the region. And of all these scores, only a small percentage has been transcribed (transcribed meaning, that the contents of the score have been annotated in any type of file, even though the most used are txt, xml or musicxml files, which will be explained with more detail in latter sections). This transcription can be done manually, but is often costly, as it involves hiring musicology experts to analyze the score's contents. Therefore, an OMR model that could identify each symbol and perfectly capture the structure of the music sheet into a file would greatly reduce the cost of music transcription.

Transcribing musical scores is of significant importance as it enables the integration of the aforementioned historical scores into modern digital applications, such as MuseScore, enhancing their accessibility to the general public. This, in turn, helps preserve and revitalize traditional Catalan music and its artists, ensuring their continued relevance in the technological era. Thus, the available transcribed musical scores can be used as data for supervised machine/deep learning models, which would benefit the research field, and in turn, get us closer to this objective.

However, one of the key issues that is keeping the field of OMR from growing is its scarcity of data. Multiple projects [16][1] have stated that the scarcity of annotated handwritten musical data is one of the biggest bottlenecks to their proposed works. Many have resorted to using printed musical data as a workaround, which can slightly increase the model's performance [12], but still, it doesn't compare to the value real handwritten data can bring to the table. A few others have spent resources on creating new datasets, a solution that is too costly to exploit. Therefore, the generation of synthetic handwritten data seems a viable option to explore.

We also must take into account that OMR is an inherently difficult field to advance, as the structure of music is more complex (2-dimensional) than, for example, the structure of text (1-dimensional). That is one of the aspects that differentiate the complexity of training OMR models from the complexity from training Handwritten Text Recognition (HTR) models. For this reason, the more data, the more the model will be able to learn the intricate structure of music.

Thus, the motivation for this work is to design an effective and robust generative architecture, in this case a GAN (which will be explained with more detail in latter sections), that can generate synthetic handwritten scores to increase the training set for OMR systems, and therefore, improve their overall performance.

For readers of this work that are not acquainted with written music, there will be a subsection in the Annex, with examples of the symbol classes more relevant to the project. Therefore whenever the reader comes across a class name (for example *AccidentalSharp*), it is recommended to take a look at this subsection for better contextualization.

2 Objectives

As stated before, the main objective of this project is to create a GAN-based architecture that can create varied and realistic images that can help improve OMR systems for handwritten scores.

However, to accomplish this goal we'll have to go through some steps or subobjectives. First of all, we will prepare a training dataset of real handwritten symbols. Then, we will design a GAN architecture that takes this dataset as input data and is capable of generating synthetic handwritten musical symbols. Once the model is trained and generates acceptable results, we will combine these synthetic symbols to create complete music lines, and finally, these lines will be tested using different metrics to evaluate their quality.

3 Methodology

Throughout this work an agile methodology has been used. This means that each time a reunion has been held, we have specified the tasks to be completed and the time needed. Once that time has passed, another meeting is held where we review the work done and decide which should be the consequent goals and the time needed before the next reunion. We believe this is the methodology most fit for this project, as it allows us to modify our objectives as the project progresses, and to catch any errors or disagreements about the work when they are still fresh.

All the work has been done in Python, using mostly *numpy, opencv* and *pillow* for image processing, *pytorch* for deep learning and the *Smashcima* [2] software for synthesizing the symbols into musical staves.

4 Planning

The initial planning of this project consisted of devising the work that should be done and dividing it into three blocks.

The first block acts as an introduction to the field and the project itself. First, multiple works on this field will be read to be well informed of the advancements already done and investigate what is there to be improved, which will serve as valuable information to write the State of the Art section of this article. Furthermore, all possible data

needed for the project will be collected and preprocessed.

The second block contains the meat of the project, as the main tasks will be designing and implementing the GAN architecture so it specializes in generating individual handwritten musical symbols. We will start by training the GAN on printed data, and when ready, it will be trained on handwritten data. Next, we will optimize hyperparameters and polish the GAN's architecture and finally, the symbols will be integrated into staves.

The third and final block encapsulates the evaluation phase, as we will analyze the quality of the synthetic data with the use of OMR-specific metrics.

Furthermore, every one or two weeks, as stated in the methodology section, a meeting has been set up to devise more specialized objectives for each 'sprint'. These can be seen in the Gantt chart below in the Annex section.

5 State of the Art

In this section, we review existing research related to the generation and recognition of handwritten symbols. We divide this review into three subsections: (1) works on Optical Music Recognition (OMR), (2) works on generation of handwritten musical symbols, and (3) projects that utilize Generative Adversarial Networks (GANs) in the broader field of Handwriten Text Recognition (HTR).

5.1 Optical Music Recognition

OMR systems rely on computer vision techniques, deep learning, and knowledge on musicology to extract symbolic music information from images. But this has not always been the case. When deep learning was not as developed as it is today, OMR works relied heavily on computer vision procedures. The work [17] by Fornés et al. is a good example, where various of these techniques can be observed, such as the use of Hough transforms to obtain the orientation of the music sheets to then rotate the image when necessary, or the use of median filters and contour tracking for staff removal. Another example is [18] by Tardón et al. where the symbol classification is done by extracting features of the image with the use of Fourier, wavelet, and angular-radial transforms and then using these features in a KNN to classify each musical symbol.

While these methods gave acceptable results, the use of deep learning models has simplified the whole pipeline of computer vision techniques that was so common in OMR, as stated in [20] by van Der Wel et al. Not only that, but deep learning techniques have also significantly improved recognition accuracy and generalization. Convolutional Neural Networks (CNNs) and Recursive Neural Networks (RNNs) have been widely used for music symbol classification and object detection within musical scores as seen in [20] by Zaragoza et al. and again, in [19] by van Der Wel et al.. As time has passed, more complex combinations of deep learning models have been developed and tested, as seen in [21] by Torras et al, where adding different language models to a baseline sequence-to-sequence model has resulted in better results.

Even with the advancements achieved throughout the latest decades, the scarcity of handwritten data is still present as stated in [1] by Torras et al., and therefore, the idea of generating synthetic yet realistic data is promising.

5.2 Generation of Handwritten Musical Symbols

As stated multiple times, the exploration of generating handwritten musical symbols is limited. The first attempts at generating new data from real samples was simple data augmentation, including techniques like rotations, mirroring and slunts. This proves effective, as demonstrated in [22] by López-Gutiérrez et al., but it still mantains numerous features from the original data, and therefore, the feature space of the augmented data is limited to a close radius from the feature space of the original data, not being able to generate totally new samples.

The creation and popularization of generative deep learning models surpassed this limitation, as it enables to generate realistic samples yet not as resembling of the input image. The leading generative models in the last decade have been Diffusion Models (DFs), Generative Adversarial Networks (GANs), AutoEncoders (AEs) and Transformers, yet not all of these have been explored in the context of handwritten OMR. Variational Autoencoders (VAEs) and Adversarial Autoencoders (AAEs) have been tested in [3] by Havelka et al. showing acceptable results. GAN architectures have also been getting attention recently as they have exhibited excellent results in other fields, as shown in the next subsection. Especifically in the field of OMR, we have works such as [23] by Shatri et al. and [24] by Tirupati et al. in from which the resulting images start to ressemble real handwritten data. Nevertheless, there is still room for progress, which is the objective of this work.

There are other types of contributions to the field of data generation that do no necessarily involve deep learning models. The work [2] by Mayer and Pecina, which will be used in this project, consists of an architecture that takes individual music symbols and inserts them into a realistic music sheet. We will be using this work to test how our synthetic music symbols, when put into sheets, react to the different metrics selected in this project.

5.3 Generation of Handwritten Text

As Handwritten Text Recognition (HTR) is a much more developed field than OMR, there's a plethora of

projects that have used GANs for generating handwritten words. Not only that, but the amount of training data (images of handwritten text) for this field is far greater than the data available for handwritten OMR (images of handwritten partitures).

First, we should mention the paper [4] by Kang et al., which, not only does it showcase a content-conditioned GAN, but it also includes a style encoder, for generating handwritten words in different author styles. This could be implemented in this project in the future once we acquire more data. Additionally, works such as [5] by Gan et al. and [6] by Luo et al., present a more complex architecture that tends to result in better performance and excellent results, as showcased in [7] by Elanwar and Betke. This presents that a GAN is a viable architecture for generating symbols, and therefore, shows the potential improvements that could be made in OMR.

6 Data Preparation and Augmentation

The first thing to do is to prepare the necessary data. For testing the GAN with printed data, we used the DeepScores V2 dataset as it allowed the GAN to train on more consistent and abundant data. We decided to use approximately 50.000 images for each symbol class and only 4 classes for the initial testing, amounting to a total of 200.000 images.

Then, we started preparing the handwritten data which ended up consisting of the following datasets:

- Muscima++ [8]
- Fornes Dataset [9]
- Homus [10]
- Capitan Collection [11]

Some manual intervention was needed as these datasets had some important differences between them. First of all, some datasets had different names for essentially the same class. This was solved via manual intervention, simply renaming the folders to the standard name from the Muscima++ dataset. The next issue was that for the means of our work, some classes needed to be divided into two separate classes. A clear example of this issue is that some types of notes were in the same class whether they were facing upwards or downwards. This is problematic, as the extreme variability of the class would make the training much more complex.

The initial class distribution is shown with blue in Figure 1, where the class with more data was *NoteheadFull*, containing over 20.000 images. This is not a bad amount, but the class distribution was highly unbalanced and it would have only been viable to work with the two more populated classes.

Thus, we decided to perform some data augmentation. First, a rotation of 10^{o} to each side was applied to every image of the dataset. An example can be seen in Figure 2. Next, we applied a flip to the images of the classes that can be flipped without losing coherence. To put some examples, the class *NoteheadFull* can be flipped horizontally and vertically, the class *cClefB* can be flipped only vertically, and the class *AccidentalFlat* can not be flipped either way. After these two augmentations, we ended up with the data distribution shown with orange in Figure 1. It can be observed that the classes have more data to work with, thus resulting in more classes being eligible to be used in our project, even though the data distribution is still a bit unbalanced. We decided to use the classes with more than 3.000 images, assuming that the other classes don't have enough training data to produce acceptable results. Thus we ended up with 49 classes.

Fig. 2 - Example of a rotation

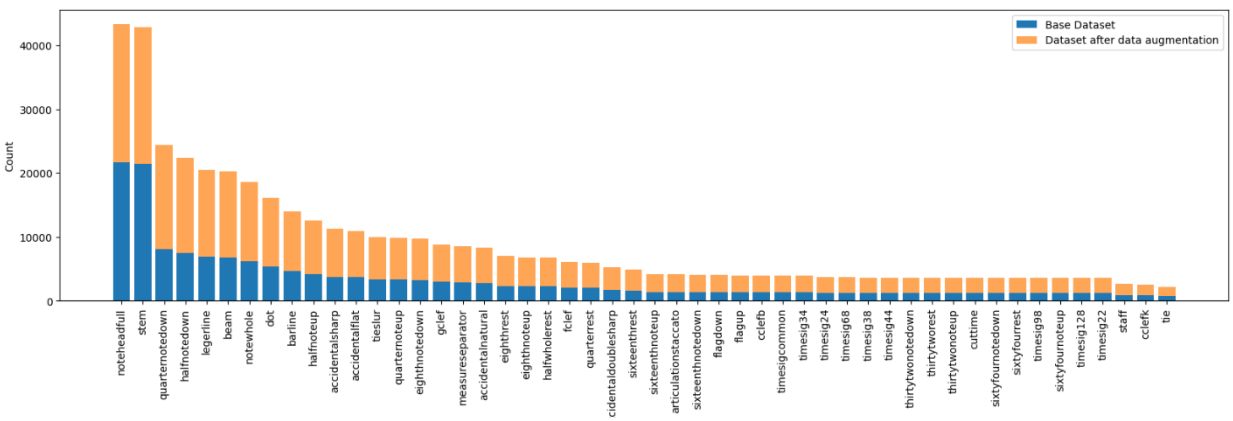


Fig. 1 - Distribution of classes before and after data augmentation

7 GAN ARCHITECTURE

Before explaining the basics of a GAN architecture, the concepts of an encoder and a decoder need to be clarified. An encoder is a neural network component that takes a structured input, such as an image or a piece of text, and transforms it into a latent and more compact representation of that input. Its inverse is a decoder, which takes the compact representation, and transforms it into a larger and structured output, in our case, an image.

Now, the architecture of a simple GAN consists of two networks: the Generator and the Discriminator. The Generator of a simple GAN, which acts as a decoder, takes a random vector and decodes it into an image. The discriminator acts as an encoder, which takes either that generated image or an image from the training data, encodes it into a set of features and from those features it tries to predict if the image is real (From the training dataset) or synthetic (Generated by the Generator). If the Discriminator gets all these predictions wrong, the Discriminator loss (which the Discriminator is trying to minimize) increases, and therefore, the Discriminator gets punished and learns from this. Instead, if the Discriminator gets all the predictions right, the Discriminator loss (which contrarily, the Generator is trying to maximize) decreases, and therefore, the Generator gets punished and learns from this. This way, these two architectures are trained simultaneously so the generator gets better at deceiving the discriminator and thus, generating images that seem more 'real' (More similar to the training data), and the discriminator gets better at identifying 'fake' from 'real' images. This min-max game has its benefits, as it encourages the Generator to constantly improve (and therefore, generate more realistic images) to deceive the Discriminator, which is also constantly improving. Nevertheless, it also has its drawbacks, the most prominent one being, that balancing the simoultaneous learning of both architectures can be difficult, and if one gets a lot superior than the other one, the inferior one won't be able to catch up, and both will get stuck as there won't be any competition. A clear visual representation of the architecture is shown in Figure 3.

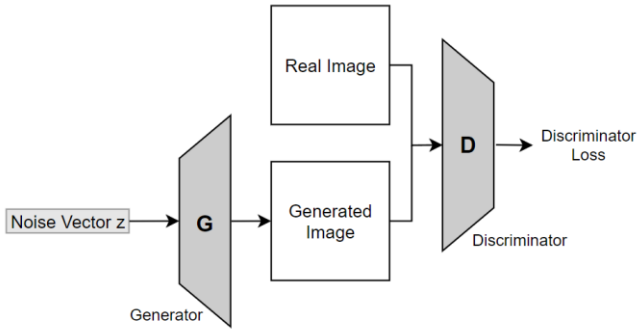


Fig. 3 - Architecture of a basic GAN

We have tweaked this basic architecture based on the improvements made in [4] by Kang et al. The first main attribute that differentiates this architecture from a simple GAN is the content conditioning. Instead of a random vector, the input of this GAN is an image of the symbol we want to generate passed through an encoder (marked as *E* in Figure 4) with the category concatenated as a one-hot-encoded vector (*O.H.E* in Figure 4). This conditions the GAN on generating the exact symbol class we want, instead of generating a random symbol from the training dataset. Also, to further influence the generation of the correct symbol, the architecture contains a music symbol identifier (*S* in Figure 4) that returns an identifier loss, which greatly punishes the Generator if it generates the wrong symbol class.

Another addition to the architecture, also based in [4] by Kang et al., is the Normal Distribution additive noise (represented in Figure 4 as *Noise Vector Z*). We do not want the generator to directly copy the symbol image passed as input, therefore, when the image is encoded into a feature vector before passing it to the generator, we sum noise from a normal distribution, so the feature vector is slightly different, and consequently, the decoded output image is also slightly different. This also helps with achieving some of the natural random variance we produce when writing the same symbol multiple times.

Our architecture and what differentiates it from a simple GAN can be observed in Figure 4. The blue box is the encoded version (therefore, the features) of the input image, the orange box is the encoded version of the category text, and the multicolor box represents both the encodings concatenated.

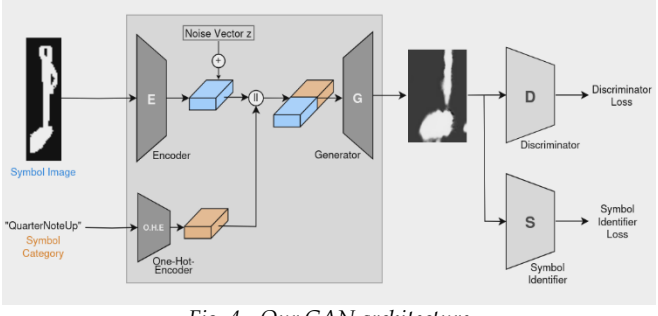


Fig. 4 - Our GAN architecture

8 EXPERIMENTS: GENERATION OF SYMBOLS

As stated before, we started with a reduced printed symbols dataset with only 4 classes. The goal of this preliminary experiment was to train the GAN with data that contains almost no variability, therefore, it would be easier to spot the major problems of the architecture. Once those were solved, we moved onto the real experiment involving the handwritten dataset, which would allow us to fix more specific problems that come with the variability of the handwritten data.

The 4 classes that were picked for the printed data are: A*ccidentalSharp*, *AccidentalFlat*, *gClef*, f*Clef*. As you can observe in Figure 5, the results are very promising. This is most likely due to the simplicity of the data, as there is no natural rotation on the symbols and therefore, the black pixels of a symbol are normally in the same area for all the images of that specific symbol.

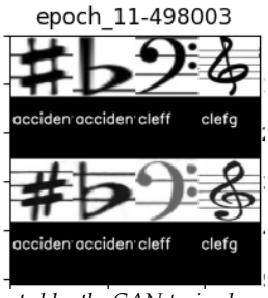


Fig. 5 - Images generated by the GAN trained on the printed symbols dataset. The upper row are the images passed as input, while the lower row are the images generated by the GAN.

After observing these results we moved onto using handwritten data.

Using the exact same GAN architecture for handwritten data did not bring the same results as using it on printed data, as can be observed in Figure 6. Our current hypothesis suggests that the variability of some symbols (In this case, the *AccidentalSharp* class) caused the GAN to not get the full picture of what the symbol should look like, and therefore, generate a blob of pixels that hardly resembles the actual symbol. The reason why it struggles with some symbols, and it does not with others is an interesting topic that can be further explored in future projects.

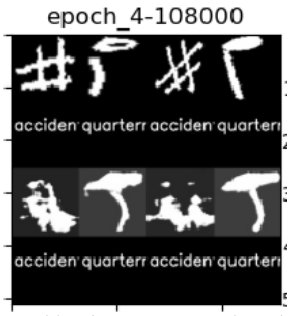


Fig. 6 - Images generated by the GAN trained on the handwritten symbols dataset. Again, the upper row are the images passed as input, while the lower row are the images generated by the GAN.

Upon re-examining Figure 6, it is evident that while the input images of quarter notes are fairly different, the resulting synthetic images are pretty similar. This indicates that the GAN is giving too much attention to the input text label (creating almost the same image for every instance of a category) and dismissing the input image that should affect the style and diversity of the symbol generated.

To address this issue we decided to randomly swap some input labels, so the GAN does not rely so much on them and pays more attention to the input image. This resulted in greater variability amongst instances of the same symbol, as can be seen in the example shown in Figure 7.

Fig. 7 - Two instances of the same symbol (halfNoteUp) in the same batch, where the generated images aren't exactly the same, as the GAN has learned some properties from the input images.

The next problem to address was the GAN's difficulty to produce high quality images for the more complicated symbols, concretely the *AccidentalSharp* and the g*Clef* classes. These results can be observed in Figure 8, where the results generated don't resemble the input symbol at all.

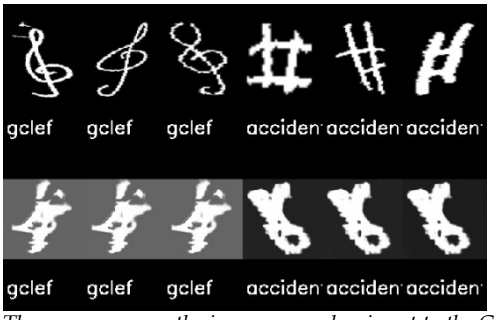


Fig. 8 - The upper row are the images passed as input to the GAN for the two more complex symbols, and the lower row are the symbols generated for each input image respectively

We have thought of two possible solutions to this problem. The first solution was to implement symbol-specific steps. That consists of, after 150 normal steps ('normal' as in every batch is filled with randomly picked symbols), the GAN does 50 steps where the batch is filled with only the symbols that prove more difficult to generate (In this case, the *AccidentalSharp* and the *gClef*). This focuses more training time on these problematic symbols.

The second solution involves the symbol classifier and its respective loss. The problem is that when these 'deformed' symbols are generated, the classifier is classifying them correctly because they resemble the real symbol to some extent (at least, more than the other classes), and therefore, there is no negative effect to the loss, thus, the GAN doesn't do anything to solve the problem. The conclusion we took from this issue is that the classifier needs to be

stricter, so it doesn't classify the deformed symbols as the real ones. For this task, we saved a lot of images of the deformed symbols and created some new 'bad' classes for the classifier (In this case, *AccidentalSharpBad* and *gClefBad*). Now, whenever an image of a deformed symbol is generated, it gets classified as an *AccidentalSharpBad* instead of an *AccidentalSharp* for example, and thus, the classifier loss is affected.

9 Automated Model Saving

Now that we have the GAN training and generating acceptable handwritten musical symbol images, we have to know when is it the best time to save the model and thus, when is it ready for production.

We could have done this manually, that is, observing the images generated during all the training of the GAN and stopping it whenever we are content with the images. But we didn't think that was the best option, first of all, because the training of the GAN can range from hours to days, and it is not efficient to take a decision based on the hundreds of thousands of images generated during two or three days. The second reason is that, if other researchers are going to use this tool in the near future, we want it to be as accessible and easy to use as possible. For these reasons, we decided to automate the saving of the model.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Fig. 9 – Euclidean distance between two points P1(x1,y1) and P2(x2,y2) in a 2D plane.

$$\cos(\Theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

Fig. 10 – Cosine similarity between two vectors A and B.

$$\mathrm{SSIM}(X, Y) = \frac{(2\mu X \mu Y + C_1)(2\sigma XY + C_2)}{(\mu^2 X + \mu^2 Y + C_1)(\sigma^2 X + \sigma^2 Y + C_2)}$$

Fig. 11 – SSIM between two images X and Y. μX and μY are the mean intensity values of each image, σ2X and σ2Y are the variances, σXY is the covariance between the two images, and C1 and C2 are constants to avoid division by 0

After some testing with different metrics, we decided to take the input images and the generated images from a batch, pass them through the encoder from our architecture, and compare these two encoded vectors using the Euclidean Distance (Figure 9) and the Cosine Similarity (Figure 10) metrics. The Euclidean distance is a measure of the straight-line distance between two points in a Euclidean space, therefore, when used with our images' feature vectors, it measures how different (thus, how far apart in feature space) are the input image and the generated image. The Cosine Similarity metric calculates the cosine of the angle between the two feature vectors, hence, unlike Euclidean Distance, that measures absolute distance, Cosine Similarity focuses more on direction rather than magnitude. The combination of these metrics yielded dull results, as the model saved neither the best generated images, nor the worst. Consequently, we also computed the SSIM (Structural Similarity Index Metric), shown in Figure 11, which is a measure that considers structural information, luminance, and contrast to better reflect human visual perception. Combined with the other two metrics, it improved performance, thus saving models that generated better images.

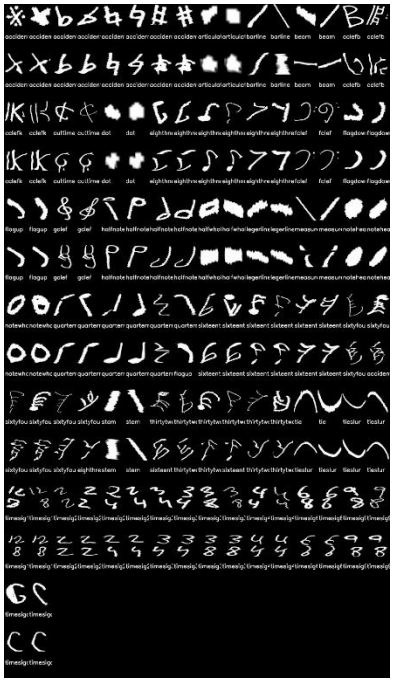


Fig. 12 - Sample image generated for each model saved

During training, the GAN saves the fifteen models that generate images with the most similarity to the input images, and to better the user experience, for each model saved, the program also saves a sample with two images generated by that model for each symbol class. That way, the user can see if a saved model really generates acceptable symbols, and because there are two images for each symbol, the user will also be able to observe variability amongst instances of the same symbol class. An example of a sample can be observed in Figure 12.

10 Sheet Generation

Once the synthetic music symbol images are generated, we need to put them into a sheets for testing. We achieved this task using the aforementioned software called Smashcima [2]. The way the base software functions is the

following. First, the user provides a musicxml file, which is a type of file with a structure similar to an xml file but specialized in storing musical information, describing the transcription and layout of the sheet we want to generate. Then, Smashcima takes information about instances of musical symbols from the Muscima++ dataset. That information includes the mask, the bounding box, and positional arguments, among others. And finally, Smashcima takes those symbols acquired from Muscima++ and places them onto a partiture sheet according to the structure represented in the musicxml file.

For our task, the second step had to be tweaked, as we need the software to take the images generated on our computer rather than all the data from the Muscima++ dataset. This was no easy task, as each symbol class had its specific parameters and therefore had to be tweaked individually.

Another problem we faced was that SMashcima produces images of whole pages of pentagrams, while the ground truth we wanted to test the results against consists of images of music sheets at line level. Thankfully the musicxml files were already divided by lines. That means that when we pass a single line musicxml file to Smashcima, it generates a whole page of staff lines where only the first one is filled with symbols. After that, we applied vertical dilation to the partiture and projected it to a 2D vector to get the distribution of the page, that way we can find the middle point between the first and the second staff so we can cut the first staff into a single image. The result can be seen in Figure 13.

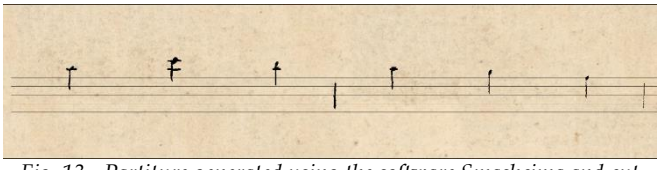


Fig. 13 - Partiture generated using the software Smashcima and cut into a single staff

Nevertheless, it is important to note that the software is still under development, and around 60% of the symbol classes generated in this project cannot be used yet.

11 RESULTS: GENERATION OF STAVES

11.1 Qualitative Results

For the qualitative results we are going to analyze first, the individual handwritten symbols generated by our GAN architecture, and then, the staves created using these symbols.

As can be seen in Figure 10, most of the symbols turned out to be recognizable and of good quality. To amend that, we have included a mix of real and synthetic symbols in section A3 of the Annex so the reader can try to guess which ones are human-generated and which are GAN-generated.

That said, there are still some symbols that our GAN architecture still struggles to generate. These symbols are mostly the *accidentalSharp and gClef*. As shown in Figures 15 and 16, even though there are some instances where the symbols generated are almost acceptable, the GAN fails to learn their inherent structure and tends to produce unrecognizable images of these symbols.

Fig. 15 – To the left, an image of an AccidentalSharp from the input dataset. To the middle and right faulty images generated by the GAN of the same symbol class

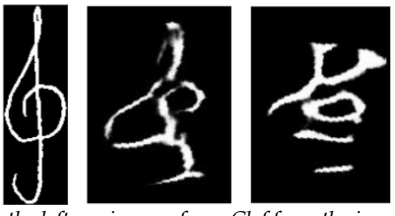


Fig. 16 – To the left, an image of an gClef from the input dataset. To the middle and right faulty images generated by the GAN of the same symbol class

Moving onto the staves generated with our synthetic symbols, the images, in some cases, could pass as written by a person using a tablet or another type of digital device. Below, in Figures 18, 19, and 20 some examples of generated staves are shown.

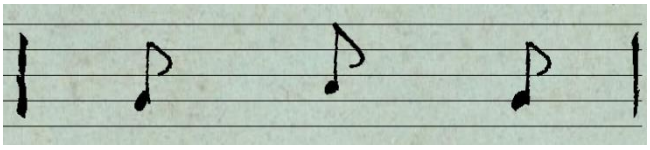


Fig. 18 - Partiture made with the symbols generated by the GAN, then cut into a measure and cropped vertically

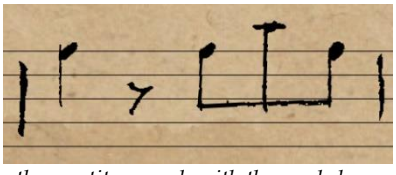


Fig. 19 – Another partiture made with the symbols generated by the GAN, then cut into a measure and cropped vertically

Nevertheless, there are still some faulty staves as seen in Figure 20. First reason being that the errors from the "symbol generation" stage transfer into the "stave generation" stage, that is, the malformed symbols generated by the GAN get put into staves making them look unpleasant and not realistic.

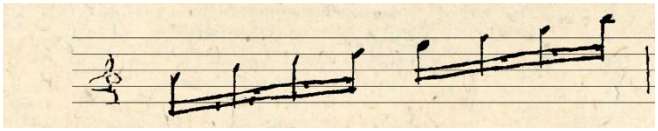


Fig. 20 – Another partiture made with the symbols generated by the GAN. This one contains one of the more faulty categories, gClef, on the most left.

The second error in this stage is that, as mentioned in the preceding section, the software used to arrange the symbols is still under development and therefore, there's still some symbol classes that are not available. Eventually, we ended up using approximately 40% of the symbol classes we generated.

11.2 Quantitative Results

Now that we have taken a look at the visual quality of the results, we will check how accurate are these synthetic symbols compared to the symbols from one of the input datasets, that is, the Muscima++.

To perform these checks, we have used the work of [25] by Pippi et al., that consists of a tool which takes two image datasets (in our case, music sheets at line level) and calculates multiple metrics between both of their latent spaces. We have decided to use the following metrics, which aim to quantify how realistically looking is the generated data.

- **Fréchet Inception Distance (FID) [14]:** It is calculated by taking two datasets, passing them through a pretrained InceptionV3 model, extracting the high-level feature embeddings from an intermediate layer of the model, and finally computing the Fréchet distance between the two distributions. During the whole process the FID assumes the data follows a Gaussian distribution.

- **Kernel Inception Distance (KID) [15]:** KID is similar to FID, in that it uses a pretrained InceptionV3 model to get the feature embeddings, but the main difference is that instead of calculating the Fréchet Distance, it calculates the Maximum Mean Discrepancy (MMD) using a polynomial kernel. Another difference is that it does not assume the feature embeddings follow a Gaussian distribution, and it tends to be more robust and stable.

- **Handwriting Distance (HWD) [13]:** A metric specifically tailored to Handwritten Text Generation (HTG). First, it extracts the features that describe a handwriting style with the use of a deep network. Then, with these features/latent space, the perceptual distance between two handwriting styles can be calculated.

  Note that in all of these metrics, the lower the value the better.

To calculate the aforementioned metrics we have used five different datasets:

- **Muscima++ Dataset [8]:** It is one of the input datasets used for the generation of synthetic symbols, and is composed of partitures written by humans with the use of a tablet. We will compare all the other datasets with this one.

- **GAN Smashcima:** This is the synthetic dataset created in this work, with the symbols generated with our GAN architecture and put into staves with the use of the Smashcima software.

- **Base Smashcima: A** dataset composed of staves generated with the base Smashcima software, which takes the musical symbols from the Muscima++ dataset.

- **Real:** A dataset composed of historical handwritten musical sheets.

- **Printed:** Images of lines from printed musical sheets, therefore, it acts as a standard of styles.

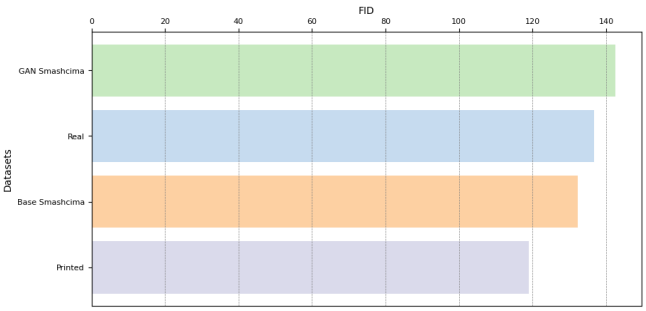


Fig. 21 – Fréchet Inception Distance (FID) between each dataset cut into lines and the Muscima++ dataset, which consists of real lines.

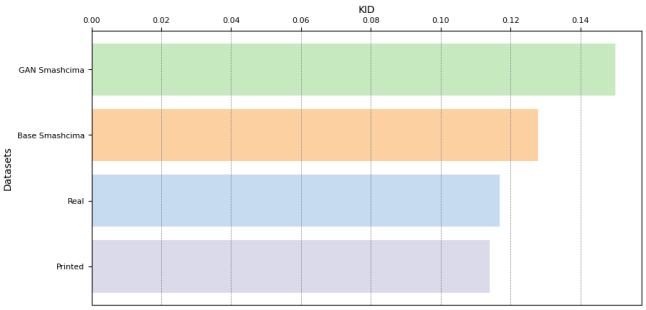


Fig. 22 – Kernel Inception Distance (KID) between each dataset cut into lines and the Muscima++ dataset, which consists of real lines.

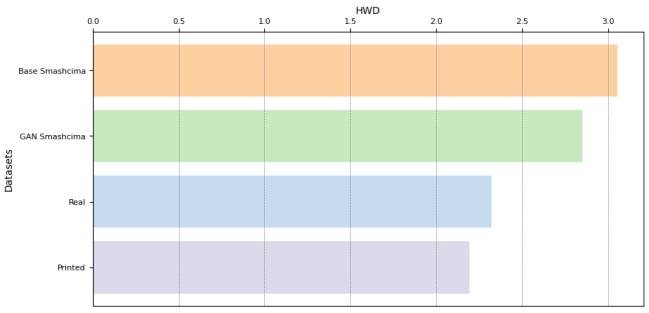


Fig. 23 – Handwriting Distance (HWD) between each dataset cut into lines and the Muscima++ dataset, which consists of real lines.

First, we can observe in Figures 21, 22, and 23 that the dataset that most ressembles Muscima++ according to all metrics is the printed one, which we expected to have a higher (worse) score, therefore, this can be an interesting topic to tackle in future research. According to the FID metric, the next most ressembling dataset is Base Smashcima, whose good result makes sense, as the symbols used in Base Smashcima are taken from the same Muscima++ dataset that it is being compared against. Next, the real dataset, also a coherent result, as it is also written by real humans. And finally, the lines generated in this work, which are really close to the FID score from the real dataset.

The results from the KID metric (Figure) are very similar, the only difference being that the Real lines scored better than the Base Smashcima ones.

And finally, the HWD metric. As seen in Figure 23, the best two results are the same as KID, that is, first the printed dataset and then the Real dataset. But the interesting thing in this chart is that the lines made with our symbols scored better in the handwritten distance metric than the base lines from Smashcima. This is unexpected because the handwriting style in Muscima++ and Base Smashcima should be very close as they use the same symbols. Nevertheless, this can be interpreted as the idea that, when no real data is available, the use of our symbols as additional synthetic data could prove effective as it, in this specific metric, has a higher resemblance to the real data compared to using just the base Smashcima software.

With this said, there is very little difference between the FID of the Real dataset and our dataset, implying that the overall structure of the image achieved by our data is pretty similar to the one from the Muscima++ dataset. Nevertheless, there is a bigger difference in the KID score, which our current hypothesis suggests that the faulty symbols generated by the GAN are the reason for this gap.

12 Conclusions and Future work

The main objective of this project has been to design a GAN architecture for generating synthetic handwritten musical symbols, a novel approach that has not been previously explored in this domain. We have demonstrated that, even in a field characterized by limited and complex data, a GAN-based model can produce promising results, offering a new avenue for data augmentation and model training in OMR.

And this pipeline opens a lot of new possibilities for future research, as, by leveraging our GAN-generated symbols, researchers can explore other techniques for arranging them into realistic sheets, or, on the other hand, different generative architectures could be tested, such as Diffusion Models or Transformers, which have not been explored in the field of OMR, and then integrated into our existing pipeline, utilizing the final processing stage to structure the generated symbols into coherent musical notation.

This said, there are some potential improvements in this work. The most important part is the imprecise generation of the three most difficult symbols mentioned throughout the project. We believe we can address this by trying to make the GAN learn the style of the input image as precisely as possible. Another problem is the lack of symbol classes that the Smashcima software accepts and includes in the generated scores. If the number of classes accepted were to be increased, we would be able to generate much more complex partitures, and therefore, be able to get closer to the feature representation of the real data.

By addressing these challenges, future iterations of this work could bring us closer to high-quality synthetic handwritten notation, facilitating data augmentation for OMR training and improving recognition systems for handwritten music. With ongoing advancements in generative models and domain adaptation techniques, we believe that the use of synthetic data in OMR will play an increasingly pivotal role in bridging the gap between handwritten and machine-readable music notation.

Acknowledgements

First of all, I want to express my gratitude to both the mentors I've had the delight to work with during this project, Alicia and Pau. I want to thank Alicia for all the guidance she has offered me and also the care for my wellbeing when I've undermined my health. I also want to thank Pau for all the tips and tricks he has taught me, and all the help he has offered to me, which I hope I haven't been too bothersome. Couldn't have wished for better mentors. Also, an extremely big thank you to Lei Kang, the master of data generation, as he has taught me a lot about this topic, and most importantly, with always a smile in his face.

Secondly, I want to thank my family for the great emotional support they have bringed me, helping me through the toughest steps of this journey. Especially, my sister Andrea, who would take me to nature, and help me lay off

temporarily of all this technology and computers.

Finally, I'm grateful for all the nice friends I've made in the CVC, which have made my job a lot more fun, and the friends from outside the CVC which have also offered entertainment and very needed relaxation of mind.

REFERENCES

[1] P. Torras, S. Biswas, and A. Fornés, "A unified representation framework for the evaluation of Optical Music Recognition systems," Int. J. Doc. Anal. Recognit. IJDAR, Jul. 2024, doi: 10.1007/s10032-024-00485-8.

[2] J. Mayer and P. Pecina, "Synthesizing Training Data for Handwritten Music Recognition," in Document Analysis and Recognition – ICDAR 2021, vol. 12823, J. Lladós, D. Lopresti, and S. Uchida, Eds., in Lecture Notes in Computer Science, vol. 12823. , Cham: Springer International Publishing, 2021, pp. 626–641. doi: 10.1007/978-3-030-86334-0_41.

[3] Havelka, J., Mayer, J., & Pecina, P. (2023, November). Symbol Generation via Autoencoders for Handwritten Music Synthesis. In *5 th International Workshop on Reading Music Systems* (p. 20).

[4] L. Kang, P. Riba, Y. Wang, M. Rusiñol, A. Fornés, and M. Villegas, "GANwriting: Content-Conditioned Generation of Styled Handwritten Word Images," Jul. 21, 2020, arXiv: arXiv:2003.02567. Accessed: Jul. 05, 2024. [Online]. Available: http://arxiv.org/abs/2003.02567

[5] J. Gan, W. Wang, J. Leng, and X. Gao, "HiGAN+: Handwriting Imitation GAN with Disentangled Representations," ACM Trans. Graph., vol. 42, no. 1, pp. 1–17, Feb. 2023, doi: 10.1145/3550070.

[6] C. Luo, Y. Zhu, L. Jin, Z. Li, and D. Peng, "SLOGAN: Handwriting Style Synthesis for Arbitrary-Length and Out-of-Vocabulary Text," Feb. 23, 2022, arXiv: arXiv:2202.11456. Accessed: Sep. 03, 2024. [Online]. Available: http://arxiv.org/abs/2202.11456

[7] R. Elanwar and M. Betke, "Generative adversarial networks for handwriting image generation: a review," Vis. Comput., Jul. 2024, doi: 10.1007/s00371-024-03534-9.

[8] J. Hajic and P. Pecina, "The MUSCIMA++ Dataset for Handwritten Optical Music Recognition," in 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto: IEEE, Nov. 2017, pp. 39–46. doi: 10.1109/ICDAR.2017.16.

[9] A.Fornés and J.Lladós and G. Sanchez, "Old Handwritten Musical Symbol Classification by a Dynamic Time Warping Based Method", in Graphics Recognition: Recent Advances and New Opportunities. Liu, W. and Lladós, J. and Ogier, J.M. editors, Lecture Notes in Computer Science, Volume 5046, Pages 51-60, Springer-Verlag Berlin, Heidelberg, 2008. DOI: 10.1007/978-3-540-88188-9_6

[10] J. Calvo-Zaragoza and J. Oncina, "Recognition of Pen-Based Music Notation: The HOMUS Dataset," 2014 22nd International Conference on Pattern Recognition, Stockholm, 2014, pp. 3038-3043. DOI: 10.1109/ICPR.2014.524

[11] Jorge Calvo-Zaragoza, David Rizo and Jose M. Iñesta. Two (note) heads are better than one: pen-based multimodal interaction with music scores. Proceedings of the 17th International Society of Music Information Retrieval conference, 2016.

[12] Baró, A., Badal, C., Torras, P., & Fornés, A. (2022, December). Handwritten Historical Music Recognition through Sequence-to-Sequence with Attention Mechanism. In *3 rd International Workshop on Reading Music Systems*

[13] V. Pippi, F. Quattrini, S. Cascianelli, and R. Cucchiara, "HWD: A Novel Evaluation Score for Styled Handwritten Text Generation," Oct. 2023.

[14] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," in Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.

[15] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, "Demystifying MMD GANs," Jan. 2021

[16] Calvo-Zaragoza, J., Jr, J. H., & Pacha, A. (2020). Understanding optical music recognition. *ACM Computing Surveys (CSUR)*, *53*(4), 1-35.

[17] A. Fornés, J. Lladós, G. Sánchez and H. Bunke, "Writer Identification in Old Handwritten Music Scores," *2008 The Eighth IAPR International Workshop on Document Analysis Systems*, Nara, Japan, 2008, pp. 347-353, doi: 10.1109/DAS.2008.29.

[18] Tardón, L.J., Sammartino, S., Barbancho, I. *et al*. Optical Music Recognition for Scores Written in White Mensural Notation. *J Image Video Proc* **2009**, 843401 (2009). https://doi.org/10.1155/2009/843401

[19] van Der Wel, E., & Ullrich, K. (2017). Optical music recognition with convolutional sequence-to-sequence models. *arXiv preprint arXiv:1707.04877*.

[20] Calvo-Zaragoza, J., Valero-Mas, J. J., & Pertusa, A. (2017, October). End-to-end optical music recognition using neural networks. In *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR* (pp. 23-27).

[21] Torras, P., Baró, A., Kang, L., & Fornés, A. (2021, November). On the Integration of Language Models into Sequence to Sequence Architectures for Handwritten Music Recognition. In *ISMIR* (pp. 690-696).

[22] López-Gutiérrez, J.C., Valero-Mas, J.J., Castellanos, F.J., Calvo-Zaragoza, J. "Data Augmentation for End-to-End Optical Music Recognition". DOI: **10.1007/978-3-030-86198-8_5**

[23] Shatri, E., Palavala, K. R., & Fazekas, G. (2024, December). Synthesising Handwritten Music with

GANs: A Comprehensive Evaluation of CycleWGAN, ProGAN, and DCGAN. In *2024 IEEE International Conference on Big Data (BigData)* (pp. 3208-3217). IEEE.

[24] Tirupati, N., Shatri, E., & Fazekas, G. (2024). Crafting Handwritten Notations: Towards Sheet Music Generation.

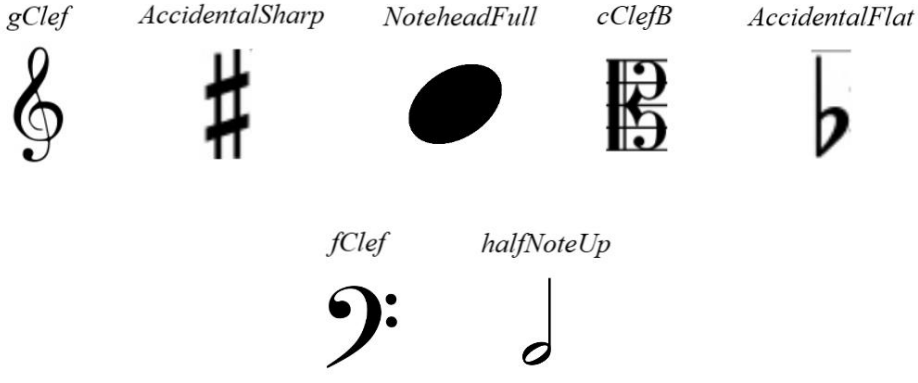
[25] Pippi, V., Quattrini, F., Cascianelli, S., & Cucchiara, R. (2023). HWD: A Novel Evaluation Score for Styled Handwritten Text Generation. *arXiv preprint arXiv:2310.20316*.

ANNEX

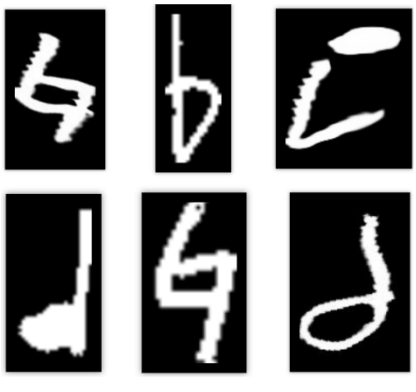
A.1 Gantt Chart

Projecte GANs	September		October				November				December				January			
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18
1st phase – Preparation																		
Define General Objectives																		
Read GAN-related papers																		
Collect and Prepare datasets																		
Apply data augmentation																		
2nd phase – Execution																		
Adapt GANWriting Architecture																		
Add Content-Conditioning Submodule																		
Change Content Recognizer to Identifier																		
Add Random Noise Injector																		
Generate Printed Images																		
Generate Handwritten Images																		
3rd phase – Image Generation																		
Polish GAN Architecture																		
Automate Model Saving																		
Generate Test Images																		
4th phase – Evaluation																		
Prepare Mashcima Software																		
Integrate Symbols into Staves																		
Prepare OMR systems																		
Test OMR systems with synthetic data																		
5th phase – Documentation																		
Add information to the report																		
Prepare presentation																		

A.2 Most Relevant Symbol classes



A.3 Guessing Game *



* Images generated by the GAN: top-left, top-right, bottom-right