Symbol Generation via Autoencoders for Handwritten Music Synthesis

Jonáš Havelka, Jiří Mayer (✉), Pavel Pecina
Institute of Formal and Applied Linguistics
Charles University, Prague, Czech Republic
Email: jonas.havelka@volny.cz, mayer@ufal.mff.cuni.cz, pecina@ufal.mff.cuni.cz
ORCID: 0000-0002-4718-3372, 0000-0001-6503-3442, 0000-0002-1855-5931

***Abstract*—Optical Music Recognition is one of the fields where synthetic data is effectively utilized for training deep learning recognition models. Due to the lack of manually annotated data, the training data is generated by an automatic procedure which produces real-looking images of music scores in large quantities. Mashcima, a system for synthesizing training data for handwritten music recognition, generates complete music scores but the individual symbols are not synthetic, they are sampled from real symbol datasets. In this paper, we explore the impact of utilizing an adversarial autoencoder within the symbol synthesis pipeline. We show that in some cases the use of an autoencoder may not only be motivated by the creation of latent-space symbol embeddings but also by improved recognition accuracy.**

***Index Terms*—Optical Music Recognition, Synthetic Training Data, Data Augmentation, Deep Learning**

I. Introduction

Synthetic training data is used in many areas of computer vision to train deep neural models, especially for tasks where obtaining sufficient amounts of manually annotated training data is prohibitively costly. This includes, e.g., handwritten text recognition [1]–[4], natural scene recognition [5], optical flow estimation [6], [7], and Optical Music Recognition (OMR) where several synthetic datasets of typeset (i.e. not handwritten) music were published recently, e.g., PrIMuS, DeepScores, DoReMi, or the Baró's dataset of historical documents [8]–[12].

For handwritten music, a synthesizer called Mashcima [13] was proposed. The current version is capable of synthesizing binarized images of singular staves of monophonic music. It exploits existing images of individual symbols and places them onto an empty staff according to a given annotation. The symbol images are taken as-is from the MUSCIMA++ dataset [15], [16]. The synthesizing process is performed in three stages as described by Mayer et al. [14]: 1) Symbol synthesis, 2) Layout synthesis, 3) Postprocessing.

In this paper, we tackle the first stage of the Mashcima's pipeline and try to improve the synthesis by generating symbol images using adversarial autoencoders (AAE) [18]. For comparison, we evaluate this approach using the same recognition model and datasets as in the original Mashcima paper [13].

This work described in this paper has been supported by the Czech Ministry of Culture (project NAKI no. DH23P03OVV008), Charles University (project GA UK no. 289623), and has been using data provided by the LINDAT/CLARIAH-CZ Research Infrastructure (https://lindat.cz), supported by the Ministry of Education, Youth and Sports of the Czech Republic (project no. LM2018101).

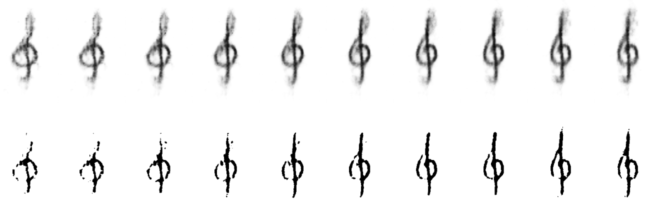
Fig. 1. Symbol interpolation using an autoencoder. Raw autoencoder output (top) and binarized using 0.5 threshold (bottom).

II. Methodology

An autoencoder is an unsupervised model that learns to compress given data to a lower-dimensional (latent) space while preserving the underlying structure of the data [21]. The model extracts useful characteristics of the data along the latent space dimensions (e.g., symbol slant, size, thickness) and then can generate symbols it has not seen in the training data, by doing interpolation in the latent space (see Figure 1).

There are many ways to build an autoencoder, utilizing convolutional networks, adversarial loss, or variational loss [18]–[21]. To limit the scope of this paper, we decided to explore adversarial autoencoders only (AAE) [18], since our early experiments showed a marginally better performance over variational autoencoders (VAE) [19].

In our experiments, we employ an AAE model with the encoder producing an L-dimensional continuous embedding vector and a C-dimensional categorical vector containing the symbol class (see Table II for the complete list). The decoder takes these two vectors and reconstructs the input image. The discriminator works only with the continuous embedding vector and forces it to have the standard normal distribution. The categorical vector is trained in a supervised fashion. Both the encoder and the decoder are convolutional (with the exception of the inner-most layer), and the discriminator is a single-hidden-layer fully-connected network. The model is described in more detail in Figure 2 and Table I.

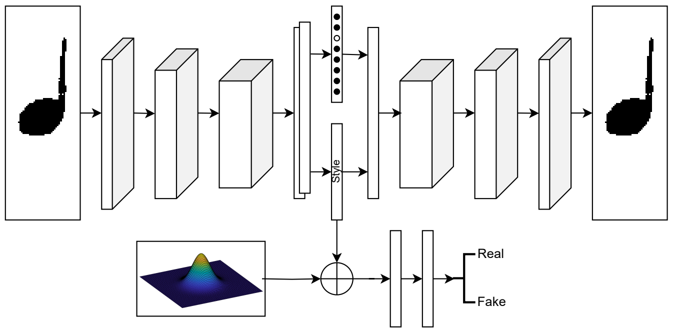

Fig. 2. Architecture of our AAE model. The input symbol (e.g., a quarter note) has its origin (notehead center) aligned with the center of the canvas. The canvas resolution is 126x459 pixels. The encoder models a Gaussian posterior, so the output is a Gaussian mean and a standard deviation vectors that are then sampled. The embedding vector has a categorical and a continuous part and the continuous part is shaped by a fully-connected discriminator network.

TABLE I
LAYERS OF THE ADVERSARIAL AUTOENCODER.

Layer	Shape	Note
Input	H × W × 1	
Convolution	H/3 × W/3 × 4	Kernel 5×5, Stride 3
Convolution	H/9 × W/9 × 16	Kernel 5×5, Stride 3
Convolution	H/27 × W/27 × 64	Kernel 5×5, Stride 3
Flatten	H/27 × W/27 × 64	
(2x) Fully connected	L + C	(one for σ, one for μ)
Normal distribution	L + C	
Latent space		
Fully connected	H/27 × W/27 × 64	
Reshape	H/27 × W/27 × 64	
Transposed conv	H/9 × W/9 × 16	Kernel 5×5, Stride 3
Transposed conv	H/3 × W/3 × 4	Kernel 5×5, Stride 3
Transposed conv	H × W × 1	Kernel 5×5, Stride 3

The discriminator takes in only the continuous embedding vector part (of size L) and applies two fully connected layers with dimensions 128 and 1 (the second one being the output layer). H and W stand for canvas width and height respectively and L and C stand for the sizes of the continuous and categorical embedding vector parts. C is equal to the number of symbol classes trained.
The last layer of both the decoder and the discriminator use the sigmoid activation function. The layer generating standard deviation (σ) has the exponential activation function, and the one generating mean (μ) has no activation function (linear). All other layers use the ReLU activation function. Each inner convolution layer (including transposed convolutions) is also followed by a batch normalization layer.

A. Symbol dataset mixing

To allow comparison with the Mashcima baseline [13], we use its subset of the MUSCIMA++ dataset as the training dataset for our autoencoder. MUSCIMA++ contains handwritten symbol images from the CVC-MUSCIMA dataset [16]. We also use the Rebelo symbol dataset [17] to expand the amount of symbols for training. The Rebelo dataset contains typeset symbols, which should help with the performance on the PrIMuS typeset evaluation dataset, as defined in the original Mashcima paper [13]. To explore all the effects, we mix these datasets in various different ways (see Figure 5) but we always generate the same amount of symbols as there are in the default Mashcima symbol repository (for each symbol category separately). These counts can be seen in the first column of Table II. These Mashcima symbols are all the available MUSCIMA++ symbols, except for those that come from writers used in the CVC-MUSCIMA evaluation dataset, as defined in the original Mashcima paper (writers 13, 17, 20, 34, 41, 49) [13].

Whenever we mix two symbol sources, we do this in a one-to-one ratio (represented by the $\bigoplus$ symbol in Figure 5). If one of the two sources does not have enough symbols, we repeatedly loop over the source. This means that symbols from the Rebelo dataset may be used multiple times. The training dataset for an autoencoder is constructed by these same rules as the final symbol repository used for the music score synthesis, unless the autoencoder is trained from a single dataset, in which case we just use the dataset as is, with no repetition nor mixing. Whenever a symbol needs to be obtained from an autoencoder, its latent space is sampled via the standard normal distribution and then decoded. It is also binarized using a 0.5 threshold, so that the synthetic image has the same appearance as the original MUSCIMA++ and Rebelo images.

B. Symbol selection and image preparation

In music notation, there are many dot-like and line-like symbols that are unsuitable for generation by convolutional neural networks. We do not synthesize these. From the rest of the music symbols, we choose to generate only those easily obtainable from the Rebelo dataset [17]. The resulting 12 symbol classes are listed in Table II.

The Mashcima synthesizer requires each symbol to have its origin point located. This is often the point that is aligned with the staff lines. MUSCIMA++ already has origins computed, but we need to do so for Rebelo as well. For quarter rests, whole notes[1], C-clefs, we choose the center of the image as the origin point. In accidentals and half notes, we find the ”hole” by traditional computer vision methods and pick its center. In the quarter and eighth notes, we find the point that’s most distant from the symbol edge. Finally, for F-clef, we choose the rightmost point of the non-dotted part situated vertically between the dots, and for G-clef, we choose the center of the most ”circular” part of the image found by the Hough circles filter [22], [23].

We align all symbols such that their origin is in the center of the image from the perspective of the autoencoder. This way, we do not need to predict the origin via regression. Notes with stems are also normalized such that the stem always points up (eighth notes are separated into two symbol classes because the flag either points left or right). We also find the tip of the

[1]Whole notes are not present in the Rebelo dataset, but we created them from half notes by removing the stem.

TABLE II
SYMBOL COUNTS IN UTILIZED SYMBOL DATASETS

Sybmol	MUSCIMA++	REBELO proc.	(REBELO)
Quarter note	15 424	634	(873)
Eighth note (up)	1 697	173	(395)
Eighth note (down)		105	
Sharp	1 689	706	(761)
Whole note	1 183	537	(—–)
Flat	1 064	544	(559)
Natural	1 021	589	(639)
Half note	845	541	(625)
Quarter rest	553	389	(389)
G-clef	341	407	(414)
F-clef	250	38	(38)
C-clef	155	130	(130)

quarter note's stem (for the synthesis of beams) as the topmost pixel of the note's image.

We do not adjust for DPI since both datasets have comparable symbol sizes (in pixels). There are some symbols from Rebelo that appear small, possibly due to a lower DPI, but it only makes the dataset more diverse and the resulting recognition model more robust. The autoencoder produces images of size 162x459 pixels, which is the smallest rectangle that fits all centered training images.

C. Autoencoder training

With the symbols prepared, the autoencoder can be trained. The training process is described in the AAE paper in the supervised section [18]. We use the Adam optimizer [24] with the learning rate 0.001 and the decays of momentum 0.9 and 0.999. We train it for 150 epochs and use batches of 50 images. Occasionally, the autoencoder has problems converging, which manifests by producing black images. We sidestepped this problem by running the experiments with multiple seeds and taking only those that converged.

D. Synthetic symbol evaluation

Once the autoencoder is trained, we use it to synthesize the final set of symbols used for music score synthesis. We modified the Mashcima code to load our own symbols instead of the MUSCIMA++ symbols. The rest of the synthesizer remained unmodified.

Mashcima uses the PrIMuS dataset for obtaining the training melodies (annotations). The second half of the annotations is randomly generated. These annotations are used with the synthetic symbols to produce the synthetic images. All of this is identical to the original version of Mashcima [13]. We use only half the available PrIMuS annotations, analogous to the experiment 3 of the paper. Our experiment A is the exact replication of the experiment 3 from the Mashcima paper.

The recognition model is a CRNN neural network, that takes in the entire staves of music and returns the corresponding annotation (a sequence of tokens in the Mashcima encoding). The evaluation is performed using Symbol Error Rate (SER), which is the number of mistakes divided by the length of the gold annotation. The number of mistakes is computed as the Levenshtein distance [25].

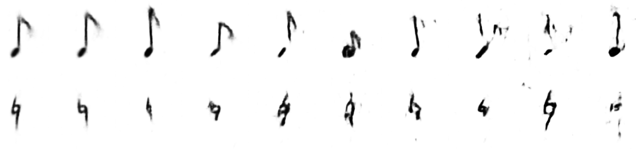
Fig. 3. Symbols sampled from the AAE with latent dimensions from left to right: 2, 2, 4, 4, 6, 6, 8, 8, 10, 10. As the dimension increases so does variability but also noisiness.

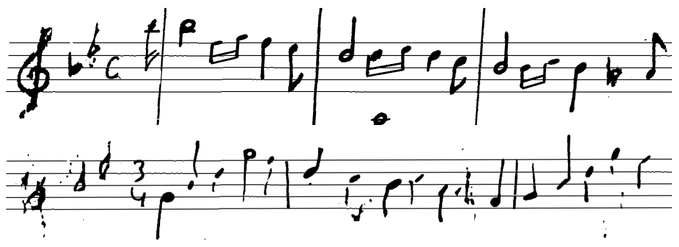
Fig. 4. Staves with symbols generated by AAE with dimension 2 (top) and 10 (bottom). The smaller dimension has prettier symbols with minimal noise. The larger dimension has symbols more noisy and some are unrecognizable (e.g. the clef). Quarter notes are also noisy, which is not the case above.

The evaluation datasets are identical to those from the Mashcima paper, namely a subset of the CVC-MUSCIMA dataset (has the same appearance as the MUSCIMA++ symbols), a subset of the typeset images from the PrIMuS dataset (has typeset appearance), and three music sheets of handwritten music titled Cavatina (referred to as *out-of-domain* test set by the original paper). This makes our experiments directly comparable with that paper.

III. EXPERIMENTS

There are many questions we want to explore. We want to know what impact has the addition of an autoencoder to the symbol synthesis. We utilize two datasets, one handwritten, one typeset. We want to explore how their combination affects the performance. We also want to find the optimal dimension of the autoencoder's latent space.

The following experiments differ in the way in which the symbol repository is prepared, but then the music image synthesis, training of the recognition model, and its evaluation are identical for all of them. Each experiment is executed 4 times, and the mean and standard deviation are computed.

A. Latent space dimension

We initially wanted to explore dimensions 2, 5, 10, 20, 50, 100, 200, 500, 1000, but experiments with dimension 20 and above never converged. The autoencoder always produced black images. Therefore, we only explore dimensions 2, 4, 6, 8, 10. We evaluate using the same setup as in experiment E in Figure 5.

By looking at the synthetic images for various latent space dimensions, one can see that smaller dimensions make the symbols better looking, but more uniform. Larger dimensions make them more diverse, but also more faulty, noisy, and less comprehensible.

TABLE III
ERROR RATE (%) FOR GIVEN AUTOENCODER LATENT SPACE DIMENSION

Dimension	CVC-MUSCIMA	PrIMuS	Cavatina
2	39.33 ± 0.47	75.10 ± 2.51	65.81 ± 1.77
4	32.05 ± 1.28	64.15 ± 2.66	64.50 ± 1.59
6	32.66 ± 0.66	58.53 ± 7.01	59.76 ± 1.60
8	30.54 ± 1.16	59.67 ± 7.20	57.22 ± 1.94
10	**30.05 ± 0.70**	**57.48 ± 1.59**	**55.96 ± 2.67**

One would expect that the presence of incomprehensible symbols would worsen the recognition model performance, but the evaluation results (Table III) show the opposite effect across all three evaluation datasets. This means that synthetic symbols need not be realistic or good-looking in order to be useful.

Based on the results, we fix the latent space dimension at 10 for all of the following experiments.

B. Topology experiments

To explore the effect of the autoencoder and each dataset, we designed a set of eight experiments. The way in which the datasets are combined, autoencoder trained, and then sampled is captured in Figure 5. Results of these experiments are present in Table IV.

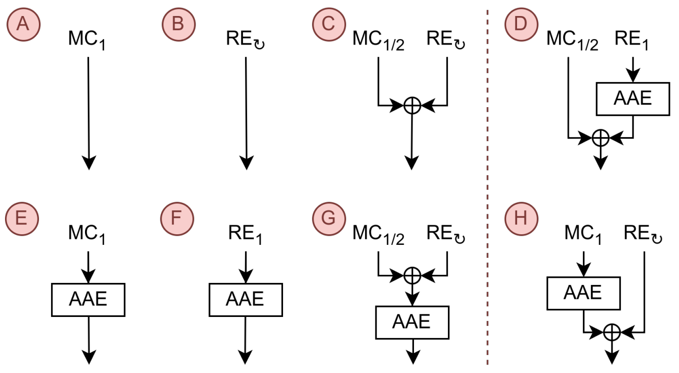

Fig. 5. Topologies of performed experiments. MC and RE are MUSCIMA++ and Rebelo symbol datasets. The subscript is the portion of the dataset used. Circular arrow means we sample the dataset more than once. Oplus combines symbols one to one. AAE is trained by incoming arrows and sampled by outgoing arrows. The final symbols used for synthesis exit the diagram at the very bottom. Looking at the experiments: going from top to bottom, we add the autoencoder; going from left to right we combine datasets in various ways. Experiments D and H are something in between so they are placed aside.

First, we consider the effect of adding an autoencoder, which corresponds to comparing experiments A-C to E-G. You can see the error rate increase on the CVC-MUSCIMA evaluation dataset, when the symbols from MUSCIMA++ are used (A,E and C,G). The same effect happens on the Cavatina test set. However, on the PrIMuS test set, the error rate decreases (for A,E and C,G). It seems that when the training and evaluation symbols come from the same domain, the autoencoder does not help, but it helps with generalization to other domains.

If we look at the experiments that use only Rebelo symbols (B,F), we see an error rate decrease for the PrIMuS test set

TABLE IV
ERROR RATES (%) FOR PROPOSED SYMBOL GENERATION TOPOLOGIES

	A	B	C	D
M:	25.02 ±1.19	39.47 ±0.83	26.98 ±1.49	**24.67 ±0.89**
	30.81 ±0.86	38.19 ±0.83	33.29 ±2.19	30.50 ±0.56
	E	F	G	H
	A	B	C	D
P:	61.17 ±6.23	77.60 ±7.84	62.63 ±6.78	53.85 ±5.21
	58.62 ±8.10	67.03 ±2.98	**52.61 ±1.24**	58.29 ±6.85
	E	F	G	H
	A	B	C	D
C:	49.65 ±1.61	57.17 ±2.56	50.17 ±2.42	**49.14 ±1.29**
	56.23 ±1.49	57.30 ±1.51	57.02 ±2.60	53.43 ±2.61
	E	F	G	H

M = CVC-MUSCIMA P = PrIMuS C = Cavatina

(and a stagnation for CVC-MUSCIMA and Cavatina). This is a decrease for the same domain evaluation. But this might be caused by the differences in the size of the MUSCIMA++ and Rebelo datasets (see Table II). For a smaller dataset, like Rebelo, the autoencoder seems to help, but for a larger dataset, like MUSCIMA++, it seems to hurt.

This hypothesis also holds when we look at the best topologies for a given test set (table numbers in bold). For CVC-MUSCIMA and Cavatina, the best result is obtained by experiment D. For PrIMuS by experiment G. The D experiment uses MUSCIMA++ without an autoencoder, but Rebelo with an autoencoder, thus avoiding the MUSCIMA++ autoencoder harm, and utilizing the Rebelo autoencoder generalization help. The G experiment utilizes one autoencoder that creates the generalization effect on Rebelo and also generalizes using MUSCIMA++ without overfitting on it (like the experiment D does).

The last thing to note is that the experiment A is the baseline from the original Mashcima paper [13], and we managed to surpass it on all three evaluation datasets (exp. D or G). Moreover, the autoencoder played an important role in this, because the simple dataset combination (exp. C) is always worse than the baseline. This indicates that the best way to incorporate smaller symbol datasets (such as Rebelo) into the synthesis pipeline is via autoencoder training.

IV. CONCLUSION

While utilizing autoencoders for music symbol synthesis is not a silver bullet, we showed that they are helpful in situations where the symbol dataset is small, or the evaluation dataset comes from a different domain than the training dataset. Another important observation is that although the synthetic symbols may look noisy they probably make the recognition model more robust. The use of autoencoders may also be motivated by other reasons, such as the use of the latent space for controlling the handwriting style. This possibility may be explored in some future work.

References

[1] Adrià Rico Blanes, “Synthetic handwritten text generation” 2018

[2] Eloi Alonso, Bastien Moysset, and Ronaldo Messina, “Adversarial Generation of Handwritten Text Images Conditioned on Sequences” 15th IAPR International Conference on Document Analysis and Recognition (ICDAR), Syndey, Australia, 2019, pp. 481-486

[3] C. V. Jawahar, and Shankar Balasubramanian, “Synthesis of Online Handwriting in Indian Languages” 10th International Workshop on Frontiers in Handwriting Recognition, La Baule, France, 2006

[4] Alex Graves (2014) “Generating Sequences With Recurrent Neural Networks” arXiv preprint arXiv:1308.0850

[5] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman (2014) “Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition” arXiv preprint arXiv:1406.2227

[6] Nikolaus Mayer, Eddy Ilg, Philipp Fischer, Caner Hazirbas, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox, “What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation?” International Journal of Computer Vision, vol. 126, pp. 942-960, 2018

[7] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox, “FlowNet: Learning Optical Flow with Convolutional Networks” IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, pp. 2758-2766

[8] Jorge Calvo-Zaragoza, and David Rizo, “End-to-End Neural Optical Music Recognition of Monophonic Scores” Applied Sciences, vol. 8, no. 4, pp. 606, 2018

[9] Jorge Calvo-Zaragoza, and David Rizo, “Camera-PrIMuS: Neural End-to-End Optical Music Recognition on Realistic Monophonic Scores” 19th International Society for Music Information Retrieval Conference (ISMIR), Paris, France, 2018, pp. 248-255

[10] Lukas Tuggener, Yvan Putra Satyawan, Alexander Pacha, Jürgen Schmidhuber, and Thilo Stadelmann, “The DeepScoresV2 Dataset and Benchmark for Music Object Detection” 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 2021, pp. 9188-9195

[11] Elona Shatri, and György Fazekas, “DoReMi: First glance at a universal OMR dataset” 3rd International Workshop on Reading Music Systems (WoRMS), Alicante, Spain, 2021, pp. 43-49

[12] Arnau Baró, Carles Badal, and Alicia Fornés, “Handwritten Historical Music Recognition by Sequence-to-Sequence with Attention Mechanism” 17th International Conference on Frontiers in Handwriting Recognition (ICFHR), Dortmund, Germany, 2020, pp. 205-210

[13] Jiří Mayer, and Pavel Pecina, “Synthesizing Training Data for Handwritten Music Recognition” 16th IAPR International Conference on Document Analysis and Recognition (ICDAR), Lausanne, Switzerland, 2021, pp. 626-641

[14] Jiří Mayer, and Pavel Pecina, “Obstacles with Synthesizing Training Data for OMR” 4th International Workshop on Reading Music Systems (WoRMS), Online, 2022, pp. 15-19

[15] Jan Hajič jr., and Pavel Pecina, “The MUSCIMA++ Dataset for Handwritten Optical Music Recognition” 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto, Japan, 2017, pp. 39-46

[16] Alicia Fornés, Anjan Dutta, Albert Gordo, and Josep Lladós, “CVC-MUSCIMA: A ground truth of handwritten music score images for writer identification and staff removal” International Journal on Document Analysis and Recognition, vol. 15, pp. 243-251, 2011

[17] Ana Rebelo, Guilherme Artur Capela, and Jaime Cardoso, “Optical recognition of music symbols”, International Journal on Document Analysis and Recognition (IJDAR), vol. 13, pp. 19-31, 2010

[18] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow (2015) “Adversarial autoencoders” arXiv preprint arXiv:1511.05644

[19] Diederik Kingma, and Max Welling, “Auto-Encoding Variational Bayes”, 2nd International Conference on Learning Representations (ICLR), Banff, Canada, 2014.

[20] Anders Boesen, Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther (2016) “Autoencoding beyond pixels using a learned similarity metric” arXiv preprint arXiv:1512.09300

[21] Dor Bank, Noam Koenigstein, and Raja Giryes (2020) “Autoencoders” arXiv preprint arXiv:2003.05991

[22] , Paul Hough (1962) “Method and means for recognizing complex patterns” patent no. US3069654A

[23] H.K. Yuen, John Princen, John Illingworth, and Josef Kittler, “Comparative study of hough transform methods for circle finding”, Image and Vision Computing, vol. 8, pp. 71–77, 1990

[24] Diederik Kingma, and Jimmy Ba, “Adam: A method for stochastic optimization”, 3rd International Conference on Learning Representations (ICLR), San Diego, USA, 2015

[25] Vladimir Levenshtein, “Binary codes capable of correcting spurious insertions and deletions of ones”, Problems of Information Transmission, 1965