






How Much Do Synthetic Datasets Matter in Handwritten Text Recognition?

Anna Wróblewska¹ , Bartłomiej Chechliński¹,
Sylvia Sysko-Romańczuk² , and Karolina Seweryn¹ 

¹ Faculty of Mathematics and Information Science,
Warsaw University of Technology, Warsaw, Poland
anna.wroblewska1@pw.edu.pl

² Management Faculty, Warsaw University of Technology, Warsaw, Poland

Abstract. This paper explores synthetic image generators in dataset preparation to train models that allow human handwritten character recognition. We examined the most popular deep neural network architectures and presented a method based on autoencoder architecture and a schematic character generator. As a comparative model, we used a classifier trained on the whole NIST set of handwritten letters from the Latin alphabet. Our experiments showed that the 80% synthetic images in the training dataset achieved very high model accuracy, almost the same level as the 100% handwritten images in the training dataset. Our results prove that we can reduce the costs of creating, gathering, and describing human handwritten datasets five times over – with only a 5% loss in accuracy. Our method appears to be beneficial for a part of the training process and avoids unnecessary manual annotation work.

Keywords: Handwritten text · Pattern recognition · Image processing · Data augmentation · Synthetic dataset · Deep learning · Autoencoder

1 Introduction

Identifying and extracting handwritten texts is still challenging in the scanning process [6]. Handwriting styles characterize by high variability across people; moreover, the inferior quality of handwritten text compared to printed text make serious difficulties when transforming it to machine-readable text. Therefore, many industries are concerned with handwritten texts, particularly healthcare and pharmaceutical, insurance, banking, and public services. Over recent years, widespread recognition technologies, like ICR (Intelligent Character Recognition) and IDR (Intelligent Document Recognition), have evolved. Recent advancements in DNN (Deep Neural Networks), such as transformer

Research was funded by the Centre for Priority Research Area Artificial Intelligence and Robotics of Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme (grant no 1820/27/Z01/POB2/2021).

architectures, have also developed progress in solving handwritten text recognition (HTR) [4]. However, models based on DNNs demand annotated datasets, which takes time and money. The motivation of the paper – to alleviate the high cost associated with annotating data to be used by machine learning algorithms – is aligned with current academic and business concerns.

The paper proposes the unsupervised generation of character image samples to alleviate the cost of manually collecting and annotating images. The proposed method consists of the following steps: (1) textually describing characters employing a set of points (coordinates) and lines linking them (straight or Bezier curves), referred to as “scheme” by the authors; (2) parsing; (3) adding some random shifts to the point coordinates and lines; (4) image rendering; (5) scaling. Then, to improve the variability of the character appearance, we used two approaches. First, the autoencoder is trained with synthetic images (generated with a schema). Then the stable points in its latent space are fixed and used to generate and variate more image samples. Second, the autoencoder is trained with handwritten images, and is then used to process the synthetic images.

In the experimental part, three types of training sets are considered: (H) only handwritten images, (S) only synthetically generated images, and (HS) a mix of handwritten images with synthetically generated images. Using training set (S) yields inferior results compared to using (H), and using training set (HS) yields results closer to those produced using (H). The experiments use a NIST hand-printed letters dataset.

We hypothesize that the model accuracy trained on the partially synthetic dataset is similar to the model trained on a fully real dataset. Nevertheless, preparing the dataset can dramatically save human labour, which significantly impacts on its business applications.

The paper contributions are:

1. synthetic character image generation using our designed schemas,
2. autoencoder-based character generation processes,
3. studying classification results for various proportions of authentic handwritten and generated images in training data.¹

In the following sections, we review relevant work on state-of-the-art methods and datasets requirements for an HTR task (Sect. 2); describe our solution for data generation (Sect. 3); outline the experimental setup that shows line of the reasoning from the dataset selection and classification training to comparison of solutions (Sect. 4); discuss the results and limitations of this study (Sect. 5); conclude and offer directions for further research (Sect. 6).

2 Related Work

The initial approaches to solving recognition challenges involved machine learning like hidden Markov models (HMM) or support vector machines (SVM) [16].

¹ Our source code for the handwritten character generator, schema examples are in: <https://github.com/grant-TraDA>.

The performance of these techniques is limited due to the manual feature extraction for inputs and their insufficient learning ability. However, RNN/LSTM can deal with sequential data (e.g. handwritten texts) to identify their patterns and generate the data. Even better results have recently been gained with multilayer perceptron neural networks [1]. Thus, sequence-to-sequence (seq2seq) models with encoder-decoder networks and attention mechanisms have gained popularity for solving recognition problems and machine translation, where output generation is essential.

The HRT challenge is to cope with large amounts of labelled data [7, 17]. Labels have to be exacted to match each character region with its name. This is done mainly by direct recognition of a word or line. For example, in many datasets, there are annotations of a word and its coordinates [18]. The character level recognition is practically tricky since the first need is to segment a word into characters. Tables 1 and 2 summarize the biggest and most popular datasets of handwritten characters, and then gather, compare, and analyze the requirements for the handwriting recognition datasets. There are synthetic datasets used for handwriting recognition training [2, 8, 9]. In [9], the authors present a vast dataset, consisting of 9 million images, created with various fonts and augmentation schemes. A similar approach was proposed by Jaderberg et al. [8], who generated the synthetic data engine that assembled data based on a few inputs – randomly chosen parameters – such as colours or fonts. Others [2] proposed a system based on GANs to create synthetic images representing handwritten words. All researchers confirm that enriching a real dataset with artificial observations improves model quality.

Table 1. Handwritten datasets.

Datasets	Description
NIST dataset [12]	Dataset published by The US National Institute of Science. It contains more than 800,000 character images from 3,600 writers. Each person completed a single page questionnaire with a few single words and one paragraph
MNIST database [10]	It is a subset of the NIST Database with only 60,000 images of handwritten digits
Devangri characters [13]	Database of handwritten Devangari characters (Indian alphabet). It contains 1,800 samples from 25 writers
Mathematics expressions [11]	Set of 10,000 expressions with 101 different mathematical symbols
Chinese characters [5]	Nearly 1,000,000 Chinese character images
Arabic printed text [15]	115,000 words images (not single characters) with 10 different Arabic fonts
Chars74K data [3]	74,000 images of digits

3 Our Method – Synthetic Dataset Generation

3.1 Synthetic Schema-Based Character Generator

The synthetic character generator is a fundamental component of our research (see Fig. 1). We provided an initial schema for each character (see the example in Fig. 2). The schema is a text file describing how a character is formed: steps that allowed the natural way of writing characters. For example, to create the character “G”, a human has to make an incomplete circle with a lot at the end. The generator loads the schema and randomly modifies it to generate the proper character. All generated characters become an input for the classifier to be trained.

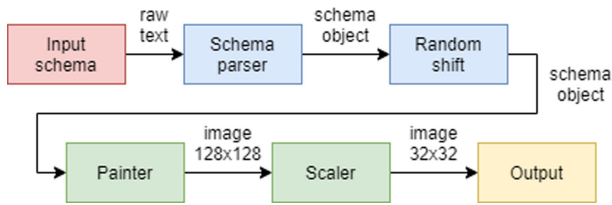


Fig. 1. Diagram of our schema-based generator. The red part was a schema file with a description of a particular character. The blue part was connected to a parser module. It loaded the schema file and translated it to the Painter, which created a character image from the schema. The yellow box represents an output character image. (Color figure online)

Schema Description. The schema is a human-readable description of how a letter should be written, e.g., making a straight line from top to bottom and adding two halves of a circle on the right side of the line. This general description was a source for all letters that followed the same handwriting. The schema contains different keywords understandable by the generator (the Parser and Painter modules) and formed a ‘standard’ character. The list below indicates primary commands to make up a character:

- *point* < name > < x > < y >
“Point” is used to create a 2D named point with “x” and “y” coordinates.
- *line* < nameA > < nameB >
The generator paints a straight line between two points points “A” and “B”.
- *bezier* < nameA > < nameB > < nameC > < nameD >.
“Bezier” creates Bezier’s curve between points “A” and “D”. Points “B” and “C” are used for deflection, so the curve does not go through them.
- *circle* < nameA > < nameB > < nameC >
Points “A”, “B” and “C” are passing by circle (or ellipse). The middle of the centre and radiuses are calculated from the points.

```

– connect_left < nameA >
  connect_right < nameA >

```

In the commands above, point “A”, was used to connect the character to the previous/next character. This keyword was used to merge subsequent letters to generate whole words. This procedure is general, and it allows for the writing of characters in any language or script.

Randomization of Generated Characters. Each schema precisely describes the process of writing only one character, so this deterministic generator created the same image each time. The characters in our datasets should imitate the human way of writing, so the characters should be randomly generated. Firstly, each point from the loaded schema was slightly randomly moved. Secondly, each line was not painted straight – some random movements approximated hand movement inaccuracy. Then, some of the pixels were erased to simulate image scanning errors. As a result, many slightly different synthetic handwritten characters were created.

An Example of a Synthetic Handwritten Character. Figure 2 presents the example of schema for generating the character “G”. The generator parses the schema and creates points (marked with red dots in Fig. 2). Points A, B, C, and D are utilised for drawing a Bezier curve [19] (B and C are used to determine the curve, A and D are the ends of the curve). Then the remainder of the straight lines is painted. Figure 2 shows the drawing process of a character with random shifts and examples of generated images.

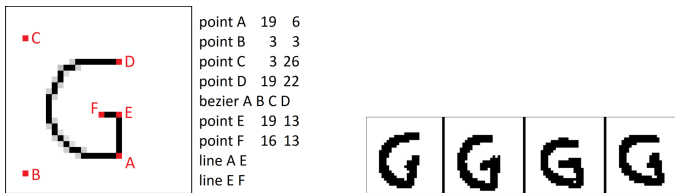


Fig. 2. Generation of a character “G”. On the left side the character with marked points and the schema text. On the right – four different characters “G” generated from the schema. (Color figure online)

A single schema provides an unlimited number of generated characters. All the characters from one schema are similar, but not the same. That allows for the preparation of massive and highly diverse training datasets for neural networks. Figure 3 presents a single sample of each character generated with the above solution.

3.2 Autoencoder-Based Generators

Many different networks can be used for a tandem of encoders and decoders. Usually, the choice is determined by the task of the autoencoder. For a character



Fig. 3. Examples of Latin alphabet characters from our generator.

generation task, we used our autoencoder: (1) as a standalone autoencoder – changing the latent representation (the encoder outputs, see Fig. 5), and (2) as a schema-based autoencoder – feeding the autoencoder with synthetic schema-based images (see Fig. 6).

Our autoencoder, convolutional-based (CNN-based) encoder-decoder neural network, was deployed on both encoder and decoder (Fig. 4). Both models operate with two kinds of convolutional filters with sizes 64 and 128. However, the order of filters in the decoder is reversed. The number and size of filters were chosen while fine-tuning. Additional extra layers were added to rescale and reshape data. Filters with smaller sizes could not have learned 25 character classes. On the other hand, increasing the size and number of filters extended computation time without visible impact on the images generated.

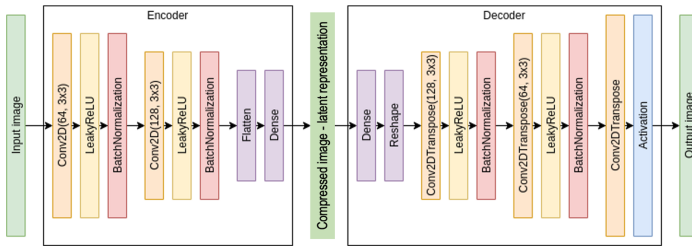


Fig. 4. Autoencoder using CNN.

Standalone Autoencoder. This approach involves only the decoder, using so-called character templates in the latent space to generate character images. The decoder should be trained with the encoder on the handwritten dataset to decode compressed characters, i.e., characters’ latent representations in the trained autoencoder. The latent representations are randomly generated with restrictions to use the shared information for each class of letters (so-called character templates, see Fig. 5).

After encoding a few previously prepared samples of each character class, the encoder output comprises a few compressed images of a single letter (their latent representations). By comparing all of the compressed samples, the templates (the latent representation) are prepared and decoded into the image of the chosen character class. Figure 5 illustrates this process: two steps in using autoencoder latent space to generate the character “K”. Encoding a few samples of “K” shows that the shared values involve eight different numbers (marked red in

Fig. 5). Other values differ between each sample. This template is the latent representation of “K”.

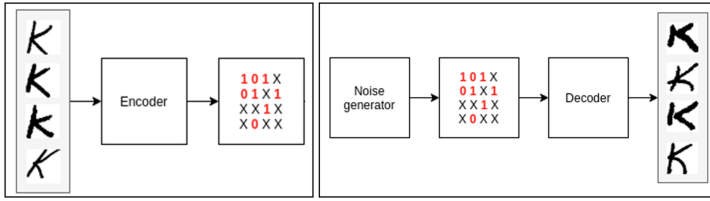


Fig. 5. Sample character generation with the usage of latent space character templates (boxes with red numbers). On the left, the process of generating latent representation templates from the input handwritten characters of the same class (here, “K” character). On the right, the inputs for the decoder during prediction (in generating images) are randomized character templates (boxes with red numbers with injected random numbers instead of crosses). (Color figure online)

The noise generator is fed with latent character representation templates and used for generating compressed data. Decoding such template data with random numbers (in place of the black crosses in Fig. 5) allows for the generation of different versions of each character class.

Schema-Based Autoencoder. This approach involves both encoder and decoder in a single module. The autoencoder trained with a dataset of authentic handwritten images was fed with the synthetic characters, i.e., those produced from our schema-based generator. Hence, the output had features more similar to human handwriting, with more randomness and noise in the output character images (compared with the solely schema-generated images). Figure 6 shows an example of processing generated images with the autoencoder, which is trained with the handwritten dataset and then fed with the schema-based generated character. After processing the image with the trained autoencoder, the character is gaining human handwriting features.

4 Research Design

Our research comprises preparation of partially synthetic handwritten characters datasets with autoencoders and schema-based generator. All the training datasets (also the proportion of real and synthetic data) is tested against the same state-of-the-art classifier architecture.

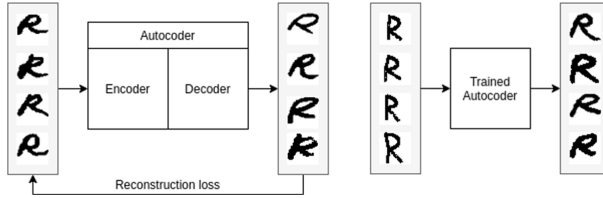


Fig. 6. The autoencoder used to process schema-based generated images. The training is done with a handwritten dataset (on the left). After the training, the schema-based characters become inputs (on the right).

4.1 Experimental Database Preprocessing

We chose NIST [12] dataset because it met all the desirable requirements (see Table 2). Hence, the remaining datasets were rejected. The NIST dataset [12] consisted of handwritten characters (pictures) made by humans (each class consisted of one character).

Table 2. Requirements for handwritten dataset comparison.

Requirement name	Description
English letters	The dataset should have all English letters. In the best case, it should contain both small and capital letters, but only capital letters will be satisfactory. Our approach can be easily extended to other scripts and languages; here, we tested English as a benchmark
Character independence	One input image should consist of one character. Character separation and concatenation are not the main goals of this benchmark research
Unified description	All input images should be prepared in the same way to avoid the unnecessary work of re-description
Number of images	The more, the better
Number of writers	The more, the better
Extras	The dataset can also have digits, non-English letters, words etc

The first layer of the CNN neural network for image recognition was designed with input neurons: one for each pixel of the input image (this is further explained in Sect. 4.3). Images from the NIST dataset, size 128×128 , required 16,384 neurons on the first layer. Therefore, memory usage and the large white background were the challenges. Empty pixels in the background were preprocessed in the following steps: (1) cropping an image to a minimal ROI (region of interest); (2) scaling it proportionally to the size of 32×32 . Image size after scaling: 32×32

or $S \times 32$ (S may vary from 32); (3) increasing the image size to 32×32 with empty pixels (white background).

4.2 Training with Synthetic and Handwritten Data

We chose samples from the experimental database to create a training and testing set in the following way. We experimented with different training datasets: (1) a handwritten train dataset, (2) a synthetic (simulated handwritten) train dataset, and (3) a mixed one. These datasets were used to train our classifier before testing on the purely handwritten dataset. The test set was always the same, consisted of the purely handwritten dataset from our database, and used after the classifier was trained to ensure stable and comparable results. The handwritten train dataset was used to compare and measure the classifier’s performance in a training environment with all images/characters written by human hands. The third mixed train dataset was changeable and depended on the test scenario: (1) synthetic images from a schema-based generator, (2) synthetic images generated with a standalone autodecoder, and (3) synthetic images generated with a schema-based generator and passed through an autoencoder and autodecoder to randomize characters.

4.3 Base Classifier for Character Recognition

VGG was selected as a classifier to train and test in different combinations for handwritten and synthetic training datasets. Currently, VGG, an object-recognition CNN that supports up to 19 layers, is one of the best models among those from the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [14]. It was designed as an improvement on AlexNet, thanks to the large kernel-size filters being replaced with multiple 3×3 kernel-sized filters [14]. In our study, we utilized VGG16 as a classifier of character images.

5 Experiments and Results

Our challenge was to evaluate how data augmentation with generated synthetic data impacts the handwritten character classification task. Table 3 presents the results.

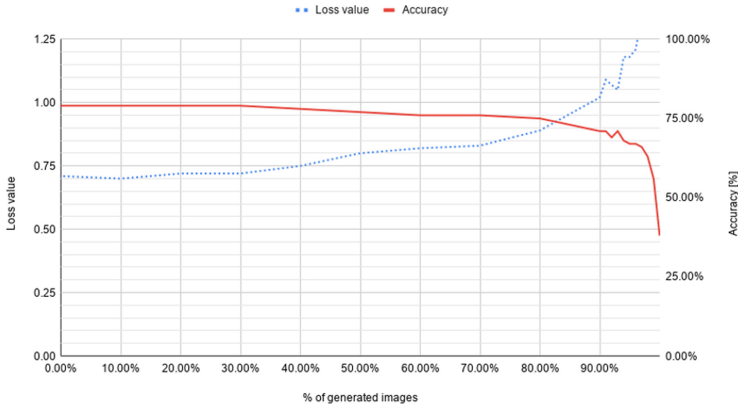
Handwritten and Synthetic Inputs. The accuracy in the training dataset with only generated inputs was around 40%, whereas solely handwritten inputs allowed for results twice as good (around 80%; see Table 3). It is worth noting that the solely synthetic images were very similar to each other, and the model got trained very fast with the early stopping algorithm (after 1–2 epochs).

Considering that, we trained our classifier on a mixed dataset, which contained both the handwritten and the generated (synthetic) images in various proportions. They were measured between each class of character images: a mixed dataset with 10% handwritten images meant that each character class

Table 3. The classification results for the designed experiments. Note: the results proceed from the test set.

Training dataset	Loss [val]	Accuracy [%]
Solely human handwritten characters	0.71	79
<i>Schema-based generator</i>		
Solely schema-based generated images	3.98	38
20% handwritten and 80% generated images	0.89	75
<i>Standalone autoencoder dataset generator</i>		
Only generated images based on autoencoder templates	3.17	39
20% handwritten and 80% generated images	0.91	77
<i>Schema-based generator followed by the autoencoder as a noise generator</i>		
Only generated images	3.11	40
20% handwritten and 80% generated images	0.89	78

had 10% handwritten images and 90% generated images in the dataset. The results on that dataset were quite surprising (see Fig. 7). The difference in accuracy between solely handwritten images and the dataset with 80% synthetic images was only four percentage points. This means that we can reduce the cost of creating, gathering, and describing human handwritten datasets five times over with only a 5% accuracy loss. The experiment proved that the assumed hypothesis about generated datasets applied in practice and brought tangible and measurable benefits.

**Fig. 7.** The classifier results for different proportions of handwritten and synthetic images in the training set. In this test, the synthetic images were produced with our schema-based generator. Note: accuracy – red line; loss value – blue dotted line. (Color figure online)

Autoencoder as Standalone Generator. We also checked how the autoencoder, as a ‘standalone’ character generator, can be applied in image generation to enrich handwritten images. Hence, letters were generated by the trained decoder with pseudo-random input noise templates achieved from the encoder outputs.

The first step was to train both encoder and decoder with the authentic handwritten dataset. When training is over, the models – the encoder and decoder – were separated. The encoder was used to get template encoded data for each class. So, the average encoded data from a few different images of the same character was calculated. This data was used to prepare inputs for the decoder. The noise generator got the templates for each class, modifying them randomly (each value is multiplied by 0.5–1.5). Such data was utilized as an input to the trained decoder. The decoder reproduced the original image from the input data. Decoded images differed because of random changes in the previous step but allowed us to quickly create a massive dataset of generated character images. In turn, the generated dataset was used to train the classifier in the same way as described in the previous tests with the original NIST dataset and noisy images from the decoder.

It was not surprising that results for the standalone autoencoder generator (see Table 3) were quite similar, but slightly better than the results gathered for schema-based generators. The classifier trained with only generated images achieved nearly 39% classification accuracy, and the mixed dataset got above 77%. The mixed dataset used 20% handwritten images and 80% characters generated with an autoencoder.

Autoencoder as Noise Generator. The subsequent test involved a schema-based generator as a source of character images. Images were generated with our schema-based generator – similar to the first test. Each image was encoded and decoded. Decoded images followed the handwriting of the trained dataset. That allowed the easy creation of a lot of different images. It is worth noting that we did not need the labelled real dataset to train the autoencoder. We needed solely handwritten characters. Finally, the generated dataset was used to train the classifier in the same way as in all our tests. The results were quite similar to the standalone autoencoder generator (see Table 3), even slightly better. It showed that the schema-based generator followed by the autocoder as a noise generator was better, as it needed less effort to create the solution.

6 Conclusions

Image generation can expand the size of existing databases at a low costs (in money and time) and bring satisfying results in sample differentiation. Additionally, image generation can be used to align datasets, where some of the classes have significantly fewer samples than others, e.g., the NIST dataset.

The schema-based solution works for the character recognition task for multiple languages and scripts (but cannot be easily applied to image processing in other tasks). The current lack of datasets that meet all the requirements for

handwritten task recognition presents an opportunity for the schema generator to create all the characters from schemes. Our results are encouraging.

Both tested autoencoder solutions need to be trained on the existing, non-labelled datasets, they can be easily applied to nearly every project with image processing. It is the most generic and the best of the tested solutions.

Our experiments confirm that inquiries into the possibilities of data augmentation by generating synthetic data, which can replace the need for authentic handwritten data to train deep learning models, is a direction researchers need to focus on.

References

1. Aaref, A.M.: English character recognition algorithm by improving the weights of MLP neural network with dragonfly algorithm. *Period. Eng. Nat. Sci.* **9**, 616–628 (2021)
2. Alonso, E., Moysset, B., Messina, R.: Adversarial generation of handwritten text images conditioned on sequences. In: *ICDAR* (2019)
3. T.E. de Campos, B.B.: Chars74k dataset (2009)
4. Chaudhuri, A., et al.: *Optical Character Recognition Systems for Different Languages with Soft Computing*. Springer, Heidelberg (2017). <https://doi.org/10.1007/978-3-319-50252-6>
5. Chen, Q.: HIT-OR3C dataset. corpus for Chinese characters (2011)
6. Geetha, R.E.A.: Effective offline handwritten text recognition model based on a sequence-to-sequence approach with CNN-RNN networks. *Neural Comput. Appl.* **33**, 10923–10934 (2021)
7. Ghosh, D., Shivaprasad, A.: An analytic approach for generation of artificial hand-printed character database from given generative models. *Pattern Recognit* **32**, 907–920 (1999)
8. Jaderberg, M., Simonyan, K., Vedaldi, A., Zisserman, A.: Synthetic data and artificial neural networks for natural scene text recognition (2014)
9. Krishnan, P., Jawahar, C.V.: Generating synthetic data for text recognition (2016)
10. LeCun, Y.: The mnist database of handwritten digits (2019)
11. Mouchère, H.: Crohme: Competition on recognition of online handwritten mathematical expressions (2014)
12. NIST: Nist handprinted forms and characters - nist special database 19 (2019)
13. Santosh, K.: Devanagari character dataset (2011)
14. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: *ICLR* (2015)
15. Slimane, F.: Arabic printed text image database (2009)
16. Taufique, M., et al.: Handwritten Bangla character recognition using inception convolutional neural network. *Int. J. Comput. Appl.* **181**, 48–59 (2018)
17. Tensmeyer, C., Wigington, C.: Training full-page handwritten text recognition models without annotated line breaks. In: *ICDAR* (2019)
18. Veit, A., Matera, T., Neumann, L., Matas, J., Belongie, S.: Coco-text: dataset and benchmark for text detection and recognition in natural images. In: *ICDAR* (2017)
19. Zhang, J.: C-Bézier curves and surfaces. *Graphical Models and Image Processing* (1999)