Generation of Synthetic Images of Full-Text Documents

Lukáš Bureš[1,2], Petr Neduchal[1,2], Miroslav Hlaváč[1,2,3], and Marek Hrúz[1]

[1] UWB, Faculty of Applied Sciences, NTIS, Pilsen, Czech Republic
{lbures, neduchal, mhruz}@ntis.zcu.cz
[2] UWB, Faculty of Applied Sciences, DEPT of Cybernetics, Pilsen, Czech Republic
{mhlavac, grubiv}@kky.zcu.cz
[3] ITMO University, St. Petersburg, Russia

Abstract. In this paper, we present an algorithm for generating images of full-text documents. Such images can be used to train and evaluate models of optical character recognition. The algorithm is modular, individual parts can be changed and tweaked to generate desired images. We describe a method for obtaining background images of paper from already digitalized documents. We use a Variational Autoencoder to train a generative model of these backgrounds enabling the generation of similar background images as the training ones on the fly. The module for printing the text uses large text corpora, font, and suitable positional and brightness noise to obtain believable results. We use Tesseract OCR to compare the real world and generated images and observe that the recognition rate is very similar indicating the proper appearance of the synthetic images. Furthermore, the mistakes made by the OCR system in both cases are alike. Finally, the system generates detailed, structured annotation of the synthesized image.

1 Introduction

Optical character recognition (OCR) is a well-developed field of computer vision and machine learning. In this paper, we present a method for digitalized text corpus generation with the focus on typewritten documents. Such documents are often targets of OCR for the purpose of data digitalization. Data digitalization is used for example for digital archiving and it enables indexing and searching for specific documents, information retrieval, or full-text search. In the past, OCR was a rule-driven system often divided into detection and recognition parts. An example of such system is Tesseract [8]. Nowadays machine learning methods such as deep neural networks (DNN) are used in an end-to-end fashion to detect and read texts [3, 2, 4, 9]. These networks are successful mainly in the domain of "wild" text detection and recognition; a task of detecting sparse text in real-world environments – meaning a lot of noise and clutter is present, and the text can be in any font/form/style. Systems of machine learning need a lot of labeled training data to learn the task and generalize it. Human labeling is a costly activity and can be faulty. For the purpose of wild text reading an algorithm for

generating synthetic training data was proposed in [1]. However, the algorithms for wild text reading are not suitable for full text documents reading. Generally, this is due to a significant difference in the appearance of the considered image data. We propose an automatic system of data generation and labeling for the purpose of training machine learning algorithms. We have access to a collection of digitalized full text documents from post World War II Czechoslovakia which we are trying to emulate.

We use several consecutive algorithms to generate believable typewritten documents. First, we automatically obtain samples of background (paper) from real documents to be able to generate empty documents. A variational autoencoder (VAE) is trained from these examples which is then able to generate reasonably looking backgrounds from random noise of given properties. Next, a text is printed onto this empty document with brightness and per-character location noise to make it look authentic. Finally, an annotation of the text is automatically generated. This process is able to generate limitless amount of text data provided a large text corpus. The generation is language independent and has several tweaking mechanisms in place.

2 Background Extraction

In this section, we describe the algorithm used for background extraction. The input of the algorithm is digital scans of original typewritten documents with dark text on a bright background. This assumption is valid not only for our data but a large set of digitalized documents. Thanks to this assumption the proposed algorithm can be based on computing the mean color.

In the first step, the input image is loaded in original colors (RGB) and in grayscale. The text is found in the grayscale image using Otsu's method [7] of brightness thresholding. In the thresholded binary image, the text pixels are stored as ones and non-text pixels as zeros. The binary image is dilated using a square window of suitable size in order to comprise pixels on the border between text and non-text pixels. These pixels are labeled as text pixels too because they can negatively affect the next step of the algorithm. Next, the mean color of the non-text pixels is computed for all color components of RGB image. The binary image from the previous step is used as a mask that removes all text pixels from the computation. All text pixels are then replaced by the mean values.

The algorithm has to be extended by one more step. The reason is that the difference between pixel values and the mean color values in particular image parts can be large – i.e the result contains brightness discontinuities which are not desired. Thus, it is necessary to re-compute the color that replaces the text color by a mean value from the text pixel's local neighborhood. All pixels are used in the local mean computation, even the already replaced text pixels. This creates a statistical bias towards the mean background color which is actually desired. This process can be viewed as a blurring using local neighborhood averaging. An example of the results of the proposed algorithm is shown in Figure 1. When there are dark artifacts in the input image, the algorithm is not able to remove them

completely. Moreover, the procedure sometimes results in high contrast noise in the output image. However, these issues are handled by the VAE generating the synthetic backgrounds and thus they need not to be addressed by this algorithm.

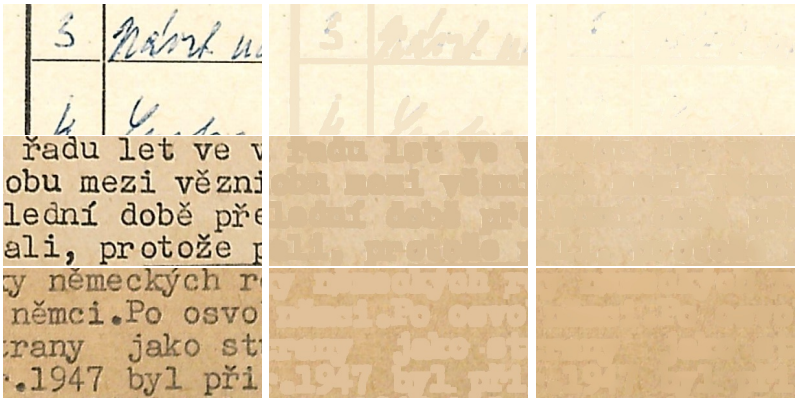

Fig. 1. Examples of extracted background regions.

3 Background Generator

In recent years, the task of generating an image from a given set of samples is handled by a special type of neural network called autoencoder. This network consists of two parts. The first part (encoder) encodes an image from a training set into a latent variable vector. The second part (decoder) then decodes the latent variable vector into the original input image. A typical autoencoder does not apply any restrictions on the latent space and thus serves as a type of memory of the training images. To further build a generative model a constraint is applied to the latent space to force it to follow a Gaussian distribution. This model is then called VAE [5]. The main disadvantage of VAE is that it produces blurry images. Various methods have been proposed to improve the sharpness of the reconstructions, most notably merging VAE with a Generative Adversarial Network [6]. We have discovered that the behavior of VAE is actually beneficial for the creation of images of old paper in the sense that it looks better. The blurry images generated by VAE eliminate the shortcomings of the Background Extraction algorithm (Section 2) by not producing the artifacts present in the training data.

Real images of 685 old paper pages were used as training data. The pages were rid of the text by the algorithm in Section 2. The original images were resized to a resolution of $128 \times 96 \times 3$ (width $\times$ height $\times$ channels). The structure of the network is described in the Table 1. The network was trained over 1000 epochs by RMSprop optimizer with the starting learning rate of 0.001. The latent space

was represented by a Gaussian with the zero mean and variance of 1.0. The size of the latent dimension of mean and variance was 250.

The output images are generated by the decoder part of the trained VAE. The input is a vector of the same size as the latent space – 250 and it is generated from a normal distribution with $mean = 0$ and $std = 0.15$. The value of the standard deviation was selected experimentally based on the analysis of the quality of the generated images. The size of generated images is $128 \times 96 \times 3$ (width $\times$ height $\times$ channels) which are then resized to the original resolution of $2480 \times 3504 \times 3$ using linear interpolation.

Table 1. Structure of VAE. The encoder is composed of four convolutional layers with 64 kernels and ReLU activation function. Every other convolutional layer is followed by batch normalization. The intermediate layer with 500 neurons has tanh activation function. The latent space is represented by two fully connected layers with 250 neurons and linear activation function.

Encoder	Decoder
64 conv(2×2), ReLU	64 deconv(3×3), ReLU
64 conv(2×2), BN, ReLU	64 deconv(3×3), ReLU
64 conv(3×3), ReLU	64 deconv(3×3), ReLU
64 conv(3×3), BN, ReLU	conv(2×2), sigmoid
500 fully-connected	

4 Synthetic Image with Text Generator

The input of our algorithm is the trained VAE model which has been described in Section 3. We use this model for generating background images on the fly. We also need a font for generating the text. We have selected a well-known font Bohemian typewriter. An example of the selected font can be seen in Figure 2. This font was used in our original dataset of scanned typewritten documents. Generally, any kind of font can be used, since the impact of the font on the proposed algorithm is minimal. We have experimentally set the font size to 45 pixels; this size has been based on the resolution of the original images and the measurements of the font in multiple randomly selected images from our private dataset.

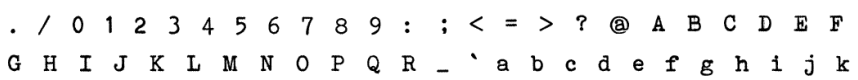

Fig. 2. An example of Bohemian typewriter font which has been used for generating synthetic documents.

For generating the synthetic images we need an input text which is loaded from a text file - this text can be anything in UTF-8, but the algorithm is limited by the selected font. When the font does not contain character which should be printed then the dummy placeholder is printed instead.

In the next step, the algorithm generates a synthetic background using the trained VAE model - we have set the resolution of the generated image to 2480×3504. This generated image has 3 channels (in RBG format). The generated background image is always unique and unpredictable in the sense of colors. We can easily change the pre-trained VAE mode to generate different kinds of background images.

For the first printing, we use an empty white image with identical resolution as the generated synthetic background image ($2480 \times 3504 \times 3$). In this white image, the algorithm prints out character by character with zero brightness. The loaded text is printed into a predefined area - the areas can be easily added and/or modified, so we are able to print the text into any predefined layout, e.g. 2 columns, 2 paragraphs, zig-zag, etc. The printing area is defined by x and y positions of the top-left corner, and width w and height h. If the text does not fit into one area, the printing process continues smoothly into the next predefined area. When there is not an area left to print into, the rest of the input text is omitted. The predefined areas have a bounded random offset which is added to x and y positions so that the generated documents look more natural.

For every character, the position is calculated based on the size of the printed character. Then the character offset generator is applied. This offset generator helps us randomly select position of a character in ± 3 pixels in x and y directions. The offset generator is in place to imitate the real old documents that have been typed with a typewriter. The offsets are set experimentally based on the resolution and size of the font.

Next, we want to simulate the brightness noise present in the original data. A vector of Gaussian noise $\mathcal{N}\left(\mu, \sigma^2\right)$ (where $\mu = 0$ and $\sigma^2 = 0.3$) is created and reshaped into an image of size width $= \frac{2480}{8}$ and height $= \frac{3504}{8}$. Then, the image is upsampled using linear interpolation into the size of the original image. This image with noise is summed with the image with printed text (only at the locations where the characters of the text are present) and the final values are clipped. In the summed image a blurring is applied with a local averaging window of size 7×7. The summed image with printed characters is blended with synthetically generated VAE background image into the final image. The image is finally blurred once again using the local averaging window of size 5×5.

We show the bounding boxes of characters in Figure 3. These are obtained by knowing several facts; the x, y position of the printed characters, the generated character offset, and the known size of the font. The size is constant since the typewriter font has a fixed width characters. We want to refine these boxes to better fit the printed characters so that the bounding box is minimal. The combined image is transformed into grayscale and contours of the characters are found. Each bounding box is then reduced to touch the contour. The words, lines, and pages bounding boxes have been adjusted too, see Figure 3.

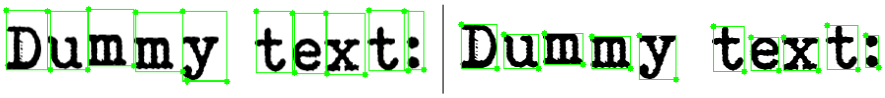

Fig. 3. The example of character's bounding boxes (left) and the example of fine–tuned character's bounding boxes (right).

All retrieved information about characters, words, lines and images is stored in files (in pickle data files and png images) for further analysis and experiments. The pickle data file contains a structure for every page. In every page there are the printing areas of defined layouts in which objects of lines are stored. All line objects contain list of words objects and finally every word object contains list of characters which the word is assembled from. Every object contains precise x and y global coordinates in original input image, and width and height of its bounding box. The information about the objects counts is stored too. This structure can be utilized in many ways. We plan to use it as annotated data for DNN learning, but it can be also used as labeled test data for already existing algorithms. Examples of the generated text can be seen in Figure 4.

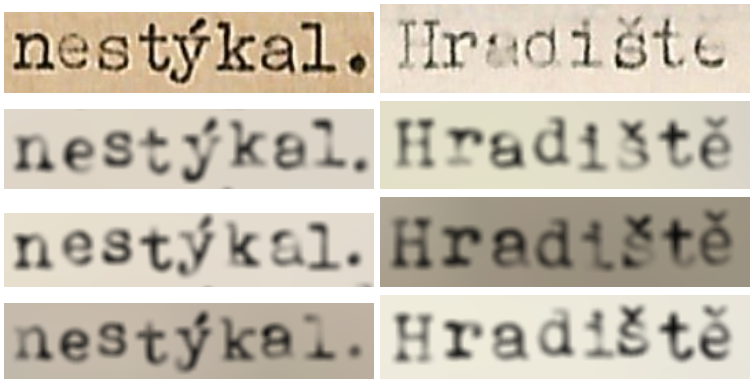

Fig. 4. Examples of generated text. Top row: original digitalized text. Next rows: generated text.

5 Experiment

The goal of our experiment is to verify whether OCR results of generated data have similar accuracy as the results of real scanned documents w.r.t. ground truth. The Tesseract system was used for the OCR task in this experiment. The accuracy w.r.t. ground truth is a distance ratio between two strings. The strings are the OCR output and the handmade ground truth text. Both are

preprocessed by erasing all characters that do not influence the meaning of the text; e.g. newlines, spaces, dots, colons etc. The accuracy is computed as follows:

$$p = \left(1 - \frac{l}{m}\right) \cdot 100 \quad [\%]\,, \tag{1}$$

where l is the Levenshtein distance between the two strings and m is the length of the longer string. In this experiment, four datasets were tested. The first one is a dataset of real documents and the other three are artificial datasets of documents generated by our proposed algorithms from the same text as is in the real dataset. Totally, there were 24 documents with 25 292 valid characters. The artificial datasets were generated with various amount of character noise – i.e. damaged the structure of printed characters in the scan (see Section 4). The results are shown in Table 2. We were able to generate dataset which has similar results as the real data as well as a dataset with significantly better accuracy result.

Table 2. OCR accuracy of real and generated scans

Dataset	Mean accuracy [%]	standard deviation
Real data	78.427	13.610
Generated 1	89.551	8.677
Generated 2	85.285	10.375
Generated 3	77.927	12.132

The results in Table 2 show that the dataset denoted Generated 3 is similarly challenging for the Tesseract system as the real data. Moreover, when observing the recognized texts, the OCR makes very similar mistakes as in the real scanned documents. The character noise used in this dataset can be used for generating benchmark datasets for comparing state of the art OCR systems. We can also control the amount of the character noise to generate less challenging data. This attribute of the algorithm can be beneficial for training DNNs. First, a DNN can be trained on easier data and then the difficulty of the data can be progressively raised. One can also generate baseline datasets that should be recognized flawlessly.

6 Conclusion and future work

We have presented an algorithm for generating images emulating real world scanned typewritten documents. The algorithm can be easily modified to generate different types of documents. The algorithm as a whole is composed of several modular algorithms. The background images representing the paper can be trained on any kind of images using a Variational Autoencoder. The font used for printing the characters can be changed easily. Layout of the page in the form of defining printing areas enables generation of arbitrary types of documents.

The character noise controls the quality of the output. The algorithm provides a detailed structured annotation of the generated documents that can be used for machine learning or testing of existing algorithms. We show experimentally that the generated documents present a similar challenge to existing OCR system as the real scanned documents.

We plan to generate huge amount of training data for deep neural networks. The performance of the trained DNN on real world data will tell us more about the quality of our proposed algorithm. There is a lot of room for tweaking different parts of the algorithm which will be our main concern in the future.

7 Acknowledgement

References

1. Gupta, A., Vedaldi, A., Zisserman, A.: Synthetic data for text localisation in natural images. In: IEEE Conference on Computer Vision and Pattern Recognition (2016)
2. Huang, W., Qiao, Y., Tang, X.: Robust scene text detection with convolution neural network induced mser trees. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision – ECCV 2014. pp. 497–511. Springer International Publishing, Cham (2014)
3. Jaderberg, M., Vedaldi, A., Zisserman, A.: Deep features for text spotting. In: European Conference on Computer Vision (2014)
4. Jaderberg, M., Simonyan, K., Vedaldi, A., Zisserman, A.: Reading text in the wild with convolutional neural networks. Int. J. Comput. Vision 116(1), 1–20 (Jan 2016), http://dx.doi.org/10.1007/s11263-015-0823-z
5. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: The International Conference on Learning Representations (2014)
6. Larsen, A.B.L., Sønderby, S.K., Larochelle, H., Winther, O.: Autoencoding beyond pixels using a learned similarity metric. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. pp. 1558–1566. ICML'16, JMLR.org (2016), http://dl.acm.org/citation.cfm?id=3045390.3045555
7. Otsu, N.: A threshold selection method from gray-level histograms. IEEE transactions on systems, man, and cybernetics 9(1), 62–66 (1979)
8. Smith, R.: An overview of the tesseract ocr engine. In: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007). vol. 2, pp. 629–633 (Sept 2007)
9. Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., Liang, J.: East: An efficient and accurate scene text detector. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 2642–2651 (2017)