Personalized Font Generator

A feature proposal for SnapChat

Renhao Liu, Washington University in St. Louis

1 Abstract

This is a proposal for a Snapchat Personalized Font Generator (PFG). The (PFG) takes an image of one's handwriting and generates digital fonts. Users may use the generated font to send messages and chat with their friends. The personalized font experience may attract more users to use the app and thus create profit for the platform. The PFG has three components: a normalizer, a generator, and a regularizer. These three parts work sequentially to produce the digital font. PFG is expected to reduce the time needed to create a font from ~8h to minutes, which makes font personalization possible in Snapchat.

Table of Contents

2 List Of Figures

3 Introduction

After the invention of printers, fonts have been formalized and restricted to their predefined shapes. Standardized fonts are easy to produce and read compared to handwritten letters. All of these features made them popular and make fonts an undeniable element for Snapchat. However, although many English fonts have been made, only about 20 fonts are most accessible to people (23 different fonts are provided in Google Doc), and only 2 - 3 are most commonly used (Like Times New Roman and Arial). In Snapchat, users can only use one font which is defined by their cell phone system.

3.1 Purpose

Recently, customized user experience like emojis and stories constantly attracts user interest and creates profits for Snapchat. Embedding custom fonts in the app could be a feature that attracts more users to Snapchat. In particular, handwritten fonts carry some features of the writer which got ignored in the printable fonts we use today. Just like voice or portraits, each person owns his/her unique handwriting styles and habits. Additionally, people could tell the writer of a document if they are familiar with the font of the writer. However, the current method of capturing fonts could take a long time (10h - 15h for English letters and common punctuation), and requires a certain degree of experience or learning in using font editing software. This report is to propose a *Font Generator* research project plan on using existing handwriting pieces to create customized fonts for Snapchat.

3.2 Current Commercial Personalized Font Solution

This section illustrates a typical commercial personalized font solution. People normally writes all the commonly used characters on a pieces of paper, scan the paper, separate out each character and use professional font edition software to adjust each character individually. This section explains conventional steps to make personalized fonts and lists out commonly used characters [7].

Steps

1. Write each character on a paper template (Figure 2.1 a)

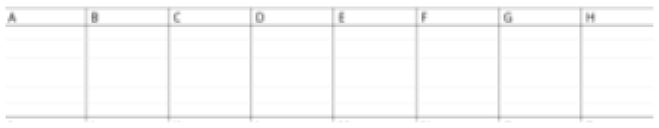


Figure 2.1 handwriting template
(from www.calligraphr.com)

2. Scan the template and extract each character
3. Adjust the size, margin, boldness of each character manually (Figure 2.2 b)

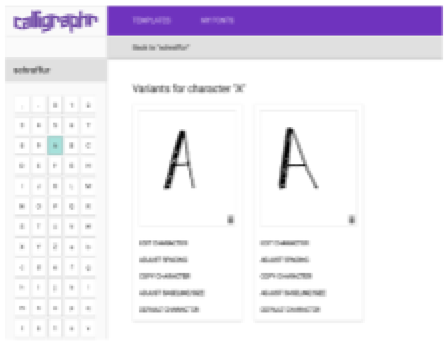


Figure 2.2. Adjustment using font software
(from www.calligraphr.com)

Characters

The American Standard Code for Information Interchange (ASCII) [4], shown in Figure 2.3, lists 93 most commonly used characters as shown in the following table (from column 2 to column 7, without SP(space) and DEL (Delete)):

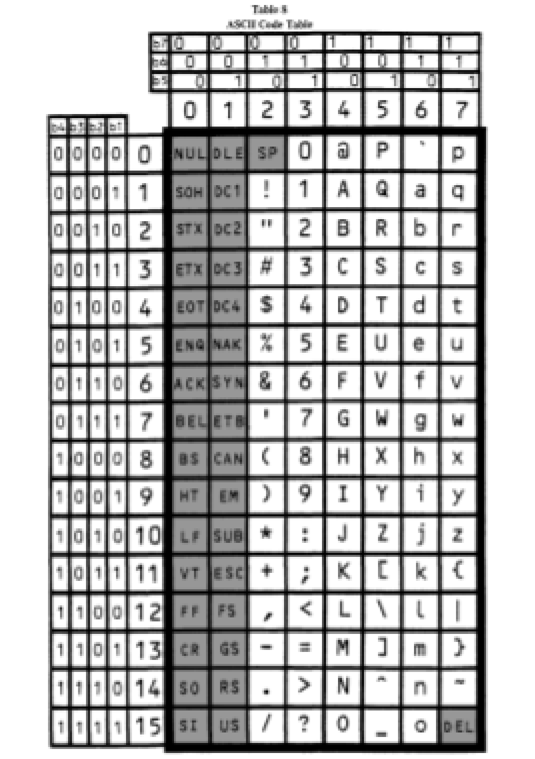


Table 8
ASCII Code Table

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
					0	1	2	3	4	5	6	7
b4	b3	b2	b1									
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Figure 2.3 ASCII Table and Commonly Used Characters

The full character set includes 93 characters mentioned above, in addition to other characters like Roman numbers and Greek letters is about 500. The whole process for creating personalized font can take several hours depending on how many characters will be included in the font and one's familiarity with font editing software. It's worth mentioning that for languages with lots of characters like Chinese or Japanese, it is almost impossible for someone to create their own font.

4 Technical Details

This section includes a detailed design of the font generator. Figure 3.1 shows the overall workflow of the font generator. Each part will be explained in detail later separately.

- *Normalizer* removes the background of an image of handwriting words and separates each character. It will recognize each hand-writing character and assign the corresponding label.
- *Generator* uses a Generative Adversarial Network to learn the writer's style and takes the learned model to generate all other characters of similar style even if they are not in the original image.
- *Reducer* aggregates all the results and produces the final product.

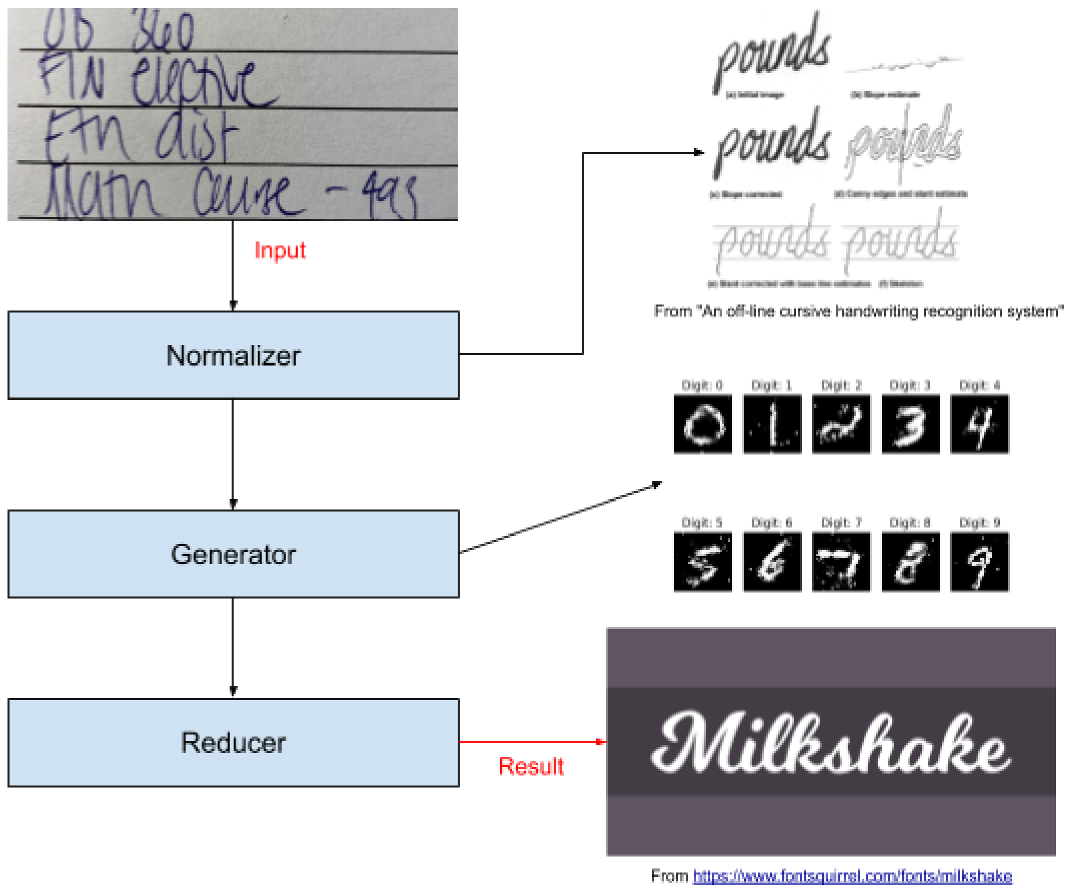


Figure 3. 1 High Level Architecture

4.1 Normalizer

The *normalizer* normalize the input image of some written text, extracts the image of individual characters from sentences and words, then recognizes what it is. The normalizer has 4 steps:

1. Background Removal
2. Words Detection
3. Normalization
4. Character recognition and labeling

(This idea partially adopts from https://github.com/Breta01/handwriting-ocr)

The first 3 steps converts the image into smaller parts with certain dimension that can be used for character detection in step 4.

Background Removal

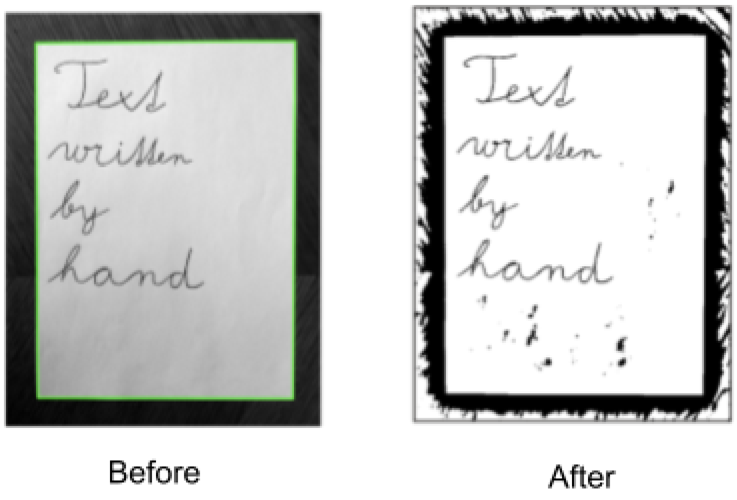


Figure 3.2 Background Removal

Background removal process (shown in Figure 3.2) removes the background of the image then adjusts the black and white value of each pixel to add contrast. The background removal process can be broken down to two steps:

1. The image is converted into a grayscale image by averaging out red, green, blue intensity of each pixel.
2. Set the darkest 10% pixels to black, and the rest to white

A Black and white (binary encoded) image is easier for text recognition in later processes as we reserve the stroke and style of each character and removes the color noise that might add unnecessary complexity to the algorithm.

Words Detection

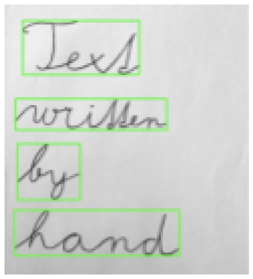


Figure 3.3 Word Detection

Word detection process (shown in Figure 3.3) breakdown the image into smaller word chunks as indicated by the green box. The process has 3 steps:

1. Taken the image from the previous step, run a breadth-first-search algorithm to group black pixels that connects to each other.
2. For each group, find the upper, lower, leftmost, and rightmost pixel to find the smallest rectangle that contains all pixels in the group.
3. Cut all of the rectangles and generate new images for the next step.

Normalization

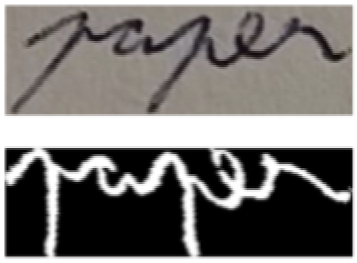


Figure 3.4 Tilt correction

The normalization process (shown in Figure 3.4) corrects tilted words and resize each word to a certain height (ex. 60 px). The normalization have 3 steps (Details discussed in Handwriting Recognition: Soft Computing and Probabilistic Approaches

8.2)

1. Create 40 alternatives of the original image with tilt range from -20 degrees to +20 degrees.
2. For each image, create a histogram along the x axis of the number of pixels, with a bucket size of 5 pixels.
3. Sum the difference between each neighbor buckets. Output the image with the max difference.
4. Scale the image to a certain height (60px) while keeping the original ratio.(We are scaling the image because The neural network used next step can only take input of constant height, other height might also work but 60px should be good enough for a character)

Character Recognition and Labeling

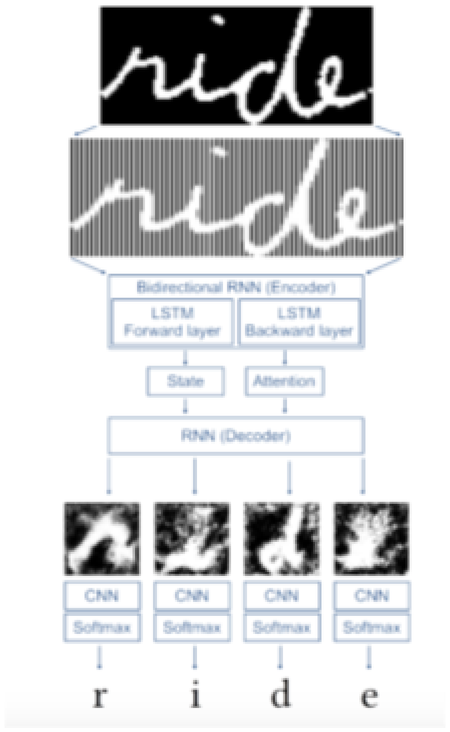


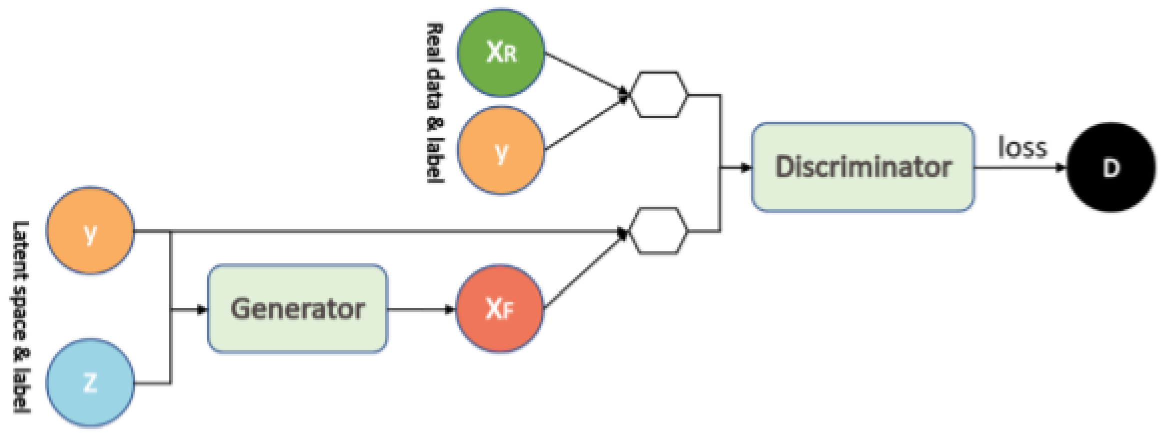
Figure 3.5 Character Recognition

The character recognition process (shown in Figure 3.5) detects each character from a word using a Bidirectional RNN encoder. Then it assigns a label to correspond to the character using a Convolutional Neural Network (CNN) and softmax. The separated characters and their labels then become the training data for the Font Generator in the next step.

4.2 Font Generator (FG)

The Font Generator generates character images that are similar to the training images. Some literature proposed style extraction on fonts [1] [3] but none of them are off-the-shelf products or does exactly what we want. So here we propose a novel way to extract font styles and generate new characters. The FG is trained with the images of characters and labels from the normalizer using a Conditional Generative Adversarial Network (CGAN) [4] as shown in Figure 4.1.

The structure of CGAN [5] is illustrated as below



(https://mc.ai/a-tutorial-on-conditional-generative-adversarial-nets-keras-implementation/)

Figure 4.1 CGAN Architecture

The CGAN has two major components: the Generator and the Discriminator (Shown in Figure 4.1). The Generator is the final product we want and will be the FG after training; The Discriminator is a helper network that helps us train the generator.

Generator

The Generator is a neural network that takes two pieces of data: a random noise input and the label we wanted. After training, it can learn the style from the font and produce a simulated image of the hand-writing character we want.

Discriminator

The discriminator is a helper neural network that gradually improves the quality of the generator during training. It takes in the real data and model and tells if it can differ the results from the generator and the real data. If the quality of the generator is poor (generates trash data), then the discriminator can easily distinguish real and poor data. Then the generator improves from the failures. At the same time, the discriminator also improves itself if it fails to differentiate real and fake data.

Steps to train a generator

1. Assign random small weights to all the edges in the network to produce random output at this point, the discriminator can easily distinguish real and fake data.
2. The generator learns from failures and produces data closer to real data.
3. The discriminator differs real and fake data and improves itself if it fails.

4. Repeat 2-3, the generator keeps improving and produces data close to real data, while the discriminator also keeps improving and distinguish the fake data from real data. (We can tell the concept of adversarial from here that the generator and discriminator fight against each other.)
5. After a certain amount of time, we take the generator and use it to get simulated images.

Prototype results

Figure 4.2 presents the result of a generator for digits 0 - 9. We can tell that the quality of hand-written digits gradually improves when we train the model for more iterations. However, we can see that the digit 2 in iteration 15000 is worse than it in iteration 10000. This is a common problem of CGAN that overtraining might hurt output quality. When to stop the algorithm will be determined in practice depend on the quality of output.

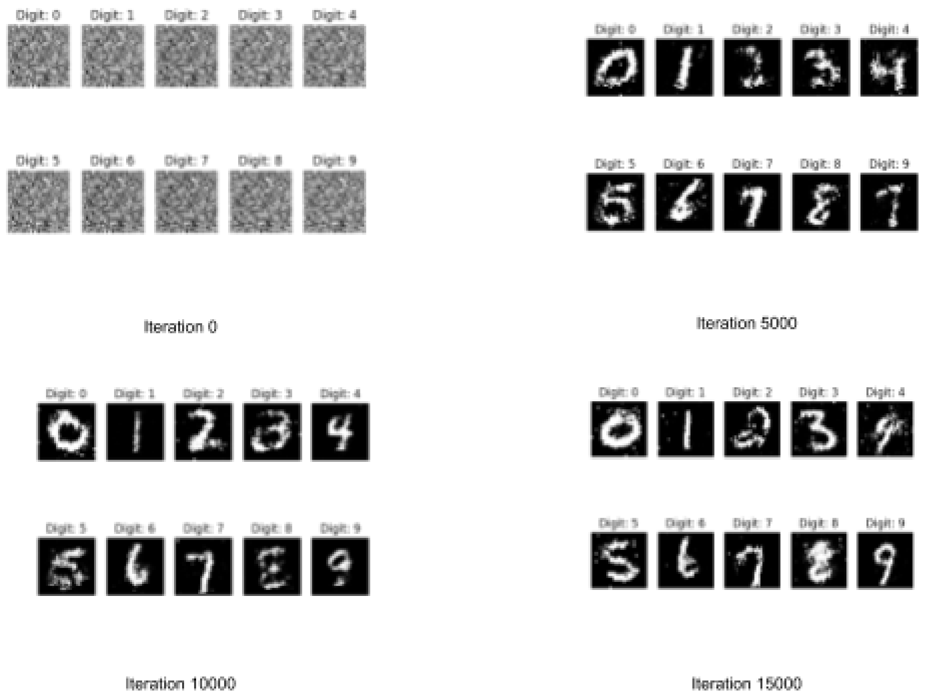


Figure 4.2 Prototype Result By Iteration

4.3 Reducer

After we train the generator, we use a normalizer to average out the noisy outputs from the generator, and convert it to the font format we normally use (OTF or TTF).

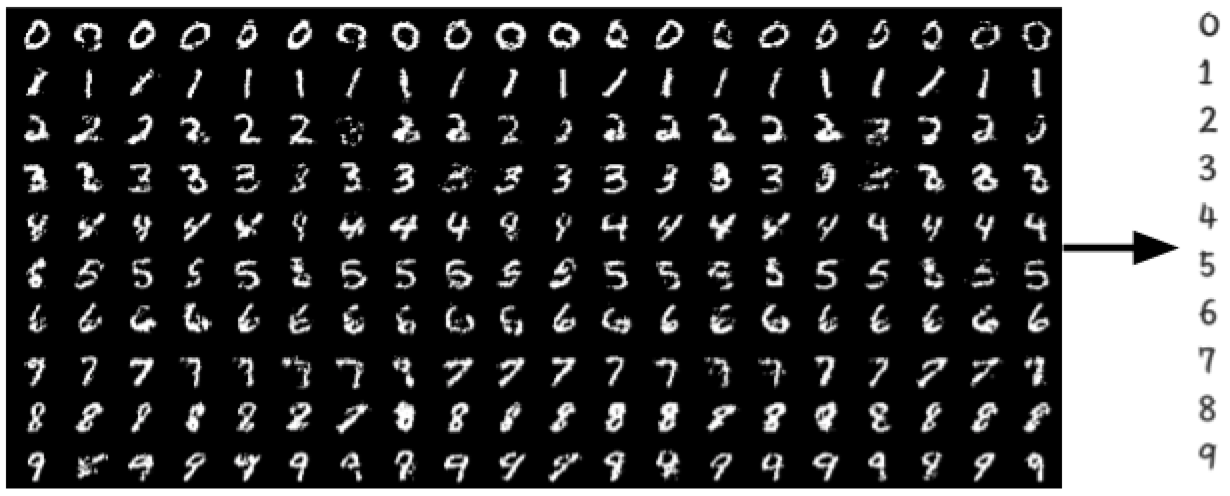


Figure 4.3 Reducer illustration

(Image from Conditional Generative Adversarial Nets [6])

Then users of Snapchat will be able to use those fonts to chat with their friends with minor update on the Snapchat client.

5 Conclusion

This font generator greatly reduces the time needed for generating personalized font from ~5 h to a couple of minutes/seconds. It has the potential to attract users in social media by providing personalized user experience.

The three components, Normalizer, Generator, and Reducer mentioned above have similar examples existed or applies simple algorithms that are easy to implement. The Normalizer takes an image of handwriting image, break it down to images of single characters with labels. Then the images and labels are used as data to train a CGAN work which allow us to generate characters with user's handwriting style. Lastly, the results from the Generator are averaged in the Reducer and converted into the digital font format to use across devices.

In all, the PFG should be applicable and may produce profit for the platform.

6 Reference

[1] Azadi, S., Fisher, M., Kim, V., Wang, Z., Shechtman, E., & Darrell, T. (2018). Multi-content GAN for Few-Shot Font Style Transfer. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. doi: 10.1109/cvpr.2018.00789

[2] Goodfellow, I. J. (2014). Generative Adversarial Nets. Retrieved from https://papers.nips.cc/paper/5423-generative-adversarial-nets

[3] Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-Image Translation with Conditional Adversarial Networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi: 10.1109/cvpr.2017.632

[4] American National Standard for Information Systems — Coded Character Sets — 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1986". American National Standards Institute (ANSI). March 26, 1986.

[5] Chm. “A Tutorial on Conditional Generative Adversarial Nets Keras Implementation.” mc.ai, June 17, 2019. https://mc.ai/a-tutorial-on-conditional-generative-adversarial-nets-keras-implementation/.

[6] Mirza, Mehdi, and Simon. “Conditional Generative Adversarial Nets.” arXiv.org, November 6, 2014. https://arxiv.org/abs/1411.1784.

[7] SAS, Spikerog. “Create Your Own Fonts.” Calligraphr. Accessed December 6, 2019. http://www.calligraphr.com/.