ScrabbleGAN: Semi-Supervised Varying Length Handwritten Text Generation Supplementary Materials

Sharon Fogel[†], Hadar Averbuch-Elor[§], Sarel Cohen, Shai Mazor[†] and Roee Litman[†]
[†] Amazon Rekognition, Israel [§] Cornell Tech, Cornell University

1. Visual Results

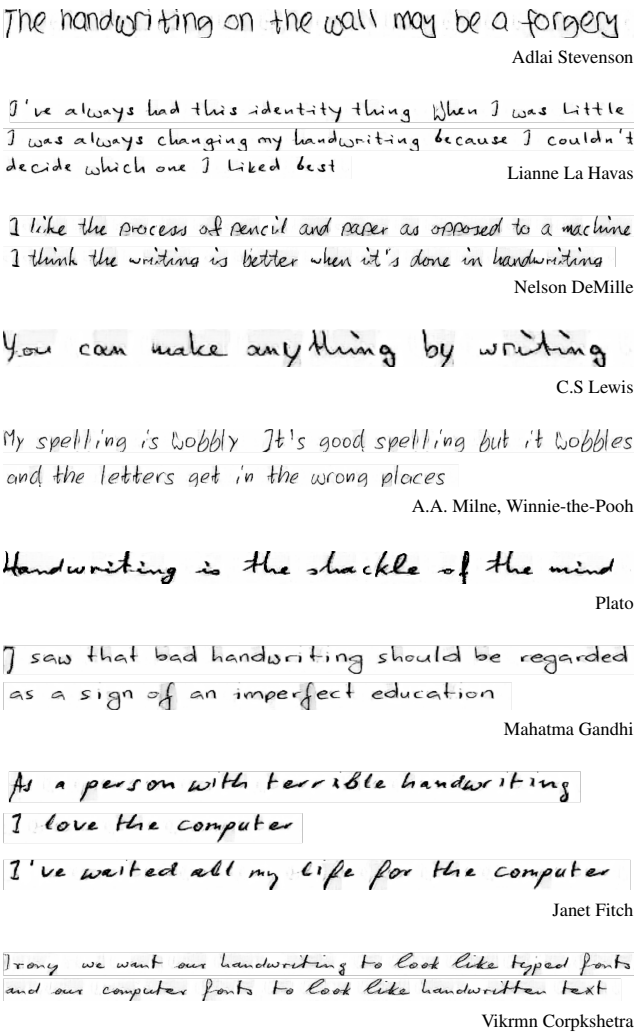

Figure 1: Quotes about handwriting. All these examples were originally one single image, and some where split into several lines to fit one column.

Generating complete sentences is one application of the varying length property of ScrabbleGAN, as can be seen in the quotes about handwriting depicted in Figure 1. Each quote was originally one single image, and was split into several lines to fit one column.

2. Ablation Study

Ablation results. In Table 1 we provide results of a few more ablation experiments, justifying selection of two more components of our framework: the architecture of $\mathcal{R}$ and the way the noise vector is fed into the network.

Modification	WER[%]	NED[%]
CNN [7]	**11.68 ± 0.29**	**3.74 ± 0.10**
CNN [7] + LSTM	13.80 ± 0.30	5.30 ± 0.13
CRNN	12.18±0.24	3.91 ± 0.08
CRNN + LSTM	12.31 ± 0.28	3.96± 0.17
ResNet + LSTM + Attn	12.27± 0.34	3.87± 0.09
CNN [7] w/o CBN [6]	12.46 ± 0.30	4.01± 0.09

Table 1: *Ablation results on genrator and recognizer architecture*, comparing HTR performance trained on different synthetic datasets. Each such set was generated by a GAN with different generator or recognizer architecture. See text for details.

Recognizer architecture selection. We tested several options for the recognizer network $\mathcal{R}$ to be used during GAN training. As mentioned in Section 3.3 in the main paper, better HTR network will not necessarily do better for ScrabbleGAN. Rows 3 through 5 in Table 1 present three alternatives from the code provided by [2]. Surprisingly, the 'weakest' configuration of the three yields the best performance, despite the fact it contains no recurrent sub network. To push this observation even further, we used a recognizer presented by [7], which contains a simple feed forward backbone of seven convolutional layers with a bidirectional LSTM on top. We tested this architecture with- and without the LSTM module, and respectively present their

performance in rows 2 and 1 of Table 1. Indeed, this simpler network helped the GAN generate the best images to be used for HTR training. Alonso el al. [1] used gated CRNN as their recognizer $\mathcal{R}$, originally presented in [3]. Since this is very similar to the CRNN presented in [2], and no implementation of [3] was provided, we chose not to include an evaluation of this specific architecture.

GAN noise input selection. As we describe in Section 3 below, we do not feed class data into CBN layers. This raised the option to remove these layer in favor of standard BN layers. As we show in the bottom row in Table 1, doing so adds about 1% to the WER score. Therefore, we opted to use CBN layers in the generator.

3. Architecture Details

We now provide some more specific implementation details for the three modules that comprise ScrabbleGAN.

Parameter	block 1	block 2	block 3
`in_channels`†	8	4	2
`out_channels`†	4	2	1
`upsample_width`	2	2	2
`upsample_height`	2	2	1
`resolution`	8	16	16
`kernel1`	3	3	3
`kernel2`	3	3	1

Table 2: Generator architecture parameters used in the helper function `G_arch` in the file `BigGAN.py`. † The number of input and output channels is the default parameter `ch=64` multiplied by the number of channels in the table.

Parameter	block 1	block 2	block 3	block 4
`in_channels`*	`input_nc`	1	8	16
`out_channels`†	1	8	16	16
`downsample`	✓	✓	✓	×
`resolution`	16	8	4	4

Table 3: Discriminator architecture parameters used in the helper function `D_arch` in the file `BigGAN.py`. * The number of input channels in the first block is the number of channels in the image (in our case 1), and in the other blocks it is the default parameter `ch=64` multiplied by the number of channels in the table. † The number of output channels is the default parameter `ch=64` multiplied by the number of channels in the table.

Generator and discriminator. We based our implementation of $\mathcal{D}$ and $\mathcal{G}$ on the PyTorch version of BigGAN [4]. The only modifications we made are in the file `BigGAN.py`. We changed the architecture parameter helpers `G_arch` and `D_arch` as described in Tables 2 and 3 respectively, in order to adjust the output patch to a size of 16×32 pixels per character. The code of the Generator class was changed accordingly to work with different width and height up-sampling parameters.

A few further modifications were made in the architecture of $\mathcal{G}$ to accommodate our scheme of class conditional generator. Unlike the original BigGAN [5] where one class is used for the entire image, here different regions of the image are conditioned on different classes (characters). Imposing this spacial condition in the first layer is easier since there is no overlap between different characters. It is more difficult, however, to feed this information directly into the CBN layers in the following blocks, due to the receptive fields overlap. For this reason, we only use the noise vectors z_2 through z_4 with no class conditioning to the CBN layers. More details about the input to the first layer appear in the implementation details section in the paper.

Recognizer. For $\mathcal{R}$ we based our implementation on the RCNN implementation by [7]. In light of the ablation presented in section 2, we decided to remove the Bi-LSTM network.

References

[1] Eloi Alonso, Bastien Moysset, and Ronaldo Messina. Adversarial generation of handwritten text images conditioned on sequences. *arXiv preprint arXiv:1903.00277*, 2019. 2

[2] Jeonghun Baek, Geewook Kim, Junyeop Lee, Sungrae Park, Dongyoon Han, Sangdoo Yun, Seong Joon Oh, and Hwalsuk Lee. What is wrong with scene text recognition model comparisons? dataset and model analysis, 2019. 1, 2

[3] Théodore Bluche and Ronaldo Messina. Gated convolutional recurrent neural networks for multilingual handwriting recognition. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 646–651. IEEE, 2017. 2

[4] Andy Brock and Alex Andonian, Biggan-pytorch, https://github.com/ajbrock/BigGAN-PyTorch, 2019-11-01. 2

[5] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 2

[6] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. 2017. 1

[7] Shu Liyang, Crnn-pytorch, https://github.com/-Holmeyoung/crnn-pytorch, 2019-11-01. 1, 2