ScrabbleGAN: Semi-Supervised Varying Length Handwritten Text Generation

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

Abstract

Optical character recognition (OCR) systems performance have improved significantly in the deep learning era. This is especially true for handwritten text recognition (HTR), where each author has a unique style, unlike printed text, where the variation is smaller by design. That said, deep learning based HTR is limited, as in every other task, by the number of training examples. Gathering data is a challenging and costly task, and even more so, the labeling task that follows, of which we focus here. One possible approach to reduce the burden of data annotation is semi-supervised learning. Semi supervised methods use, in addition to labeled data, some unlabeled samples to improve performance, compared to fully supervised ones. Consequently, such methods may adapt to unseen images during test time.

We present ScrabbleGAN, a semi-supervised approach to synthesize handwritten text images that are versatile both in style and lexicon. ScrabbleGAN relies on a novel generative model which can generate images of words with an arbitrary length. We show how to operate our approach in a semi-supervised manner, enjoying the aforementioned benefits such as performance boost over state of the art supervised HTR. Furthermore, our generator can manipulate the resulting text style. This allows us to change, for instance, whether the text is cursive, or how thin is the pen stroke.

1. Introduction

Documentation of knowledge using handwriting is one of the biggest achievements of mankind: the oldest written records mark the transition from prehistory into history, and indeed, most evidence of historic events can be found in handwritten scripts and markings. Handwriting remained the dominant way of documenting events and data well after Gutenberg's printing press in the mid-1400s. Both printing and handwriting are becoming somewhat obsolete in the digital era, when courtroom stenographers are being replaced by technology [4], further, most of the text we type remains in digital form and never meets a paper.

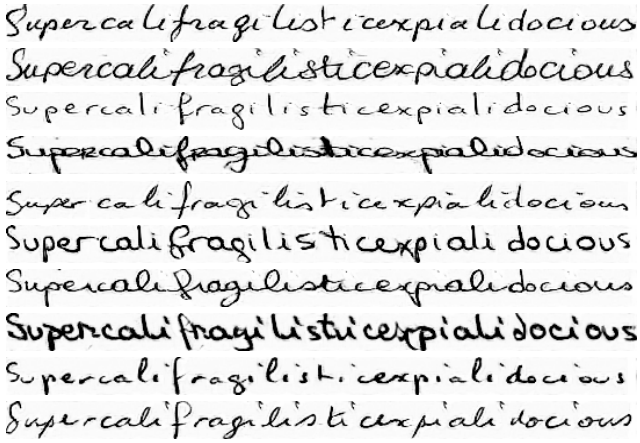


Figure 1: The word "Supercalifragilisticexpialidocious" (34 letters) from the movie "Mary Poppins" written in different styles using our network. Note that some of these styles are cursive.

Nevertheless, handwritten text still has many applications today, a huge of amount of handwritten text has accumulated over the years, ripe to be processed, and still continues to be written today. Two prominent cases where handwriting is still being used today are healthcare and financial institutions. There is a growing need for those to be extracted and made accessible, e.g. by modern search engines. While modern OCRs seem to be mature enough to handle printed text [20, 21], *handwritten text recognition* (HTR) does not seem to be on par. We attribute this gap to both the lack of versatile, annotated handwritten text, and the difficulty to obtain it. In this work, we attempt to address this gap by creating real-looking synthesized text, reducing the need for annotations and enriching the variety of training data in both style and lexicon.

Our contributions are threefold; First, we present a novel fully convolutional handwritten text generation archi-

Corresponding author: shafog@amazon.com
[§] work done while working at Amazon.

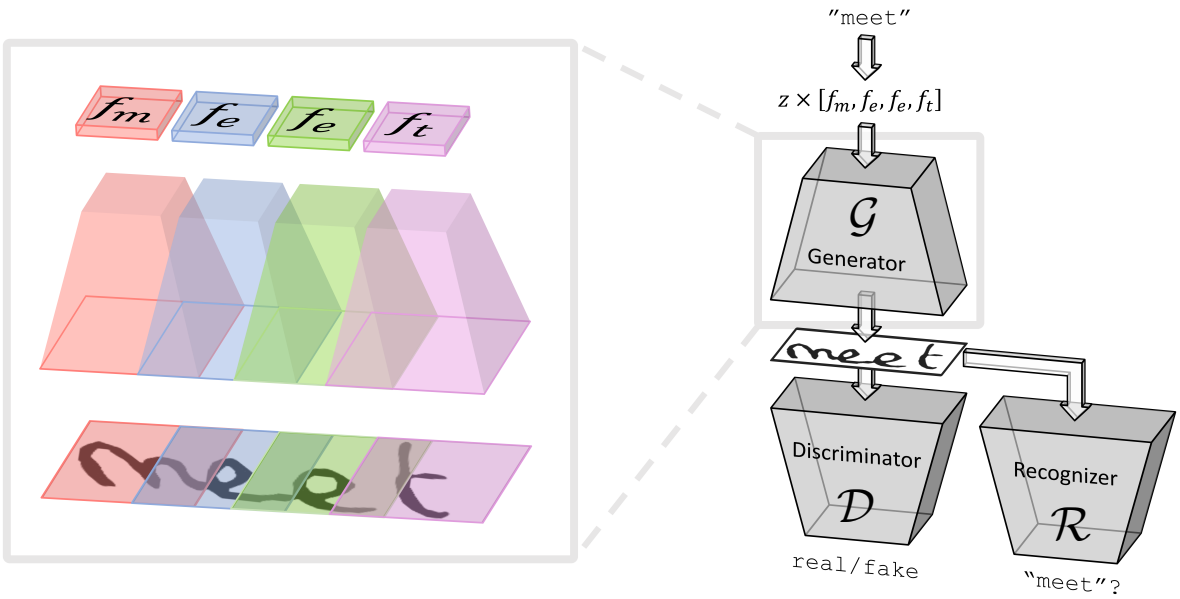


Figure 2: *Architecture overview* for the case of generating the word "meet". **Right**: Illustration of the entire ScrabbleGAN architecture. Four character filters are concatenated (f_e is used twice), multiplied by the noise vector z and fed into the generator $\mathcal{G}$. The resulting image is fed into both the discriminator $\mathcal{D}$ and the recognizer $\mathcal{R}$, respectively promoting style and data fidelity. **Left**: A detailed illustration of the generator network $\mathcal{G}$, showing how the concatenated filters are each fed into a class-conditioned generator, where the resulting receptive fields thereof are overlapping. This overlap allows for adjacent characters to interact, enabling cursive text, for example.

tecture, which allows for arbitrarily long outputs. This is in contrast to the vast majority of text related solutions which rely on recurrent neural networks (RNN). Our approach is able to generate arbitrarily long words (e.g., see Figure 1) or even complete sentences altogether. Another benefit of this architecture is that it learns character embeddings without the need for character level annotation. Our method's name was chosen as an analogy between the generation process to the way words are created during the game of Scrabble, i.e. by *concatenating* some letter-tokens together into a word. Second, we show how to train this generator in a semi-supervised regime, allowing adaptation to unlabeled data in general, and specifically to the test time images. To the best of our knowledge, this is the first use of *unlabeled* data to train a handwritten text synthesis framework. Finally, we provide empirical evidence that the training lexicon matters no less than the richness of styles for HTR training. This fact emphasizes the advantage of our method over ones that only warp and manipulate the training images.

2. Previous Work

Handwriting text recognition can be seen as a specific case of *optical character recognition* (OCR). This is a well studied topic, in the in-depth survey [34], HTR approaches are divided into online and offline methods, which differ by the type of data they consume: Online methods have access to the pen location as the text is being written, and hence can disambiguate intersecting strokes. Offline methods, conversely, have access only to the final resulting text image (i.e. rasterized), possibly also in the presence of some background noise or clutter. Clearly, online methods have a strict advantage over their offline counterparts in terms of data quality, but require additional equipment (such as a touchscreen) to capture pen stroke data. Hence, online data is harder to create in large quantities, especially in a natural setting. Furthermore, these methods are unsuitable for historic manuscripts and markings which are entirely offline. For this reason, we chose to focus on offline methods and leave online methods out of the scope for this manuscript.

Modern HTR methods harness the recent advancements in deep networks, achieving top performance on most, if not all, modern benchmarks. Many of these methods are inspired by the *convolutional recurrent neural network* (CRNN) architecture, used originally for scene text recognition by Shi et al. [38]. Poznanski et al. [35] used a CNN to estimate the n-grams profile of an image and match it to the profile of an existing word from a dictionary. PHOCNet by Sudholt et al. [39] extended the latter by employing a pyramidal histogram of characters (PHOC), which was used mainly for word spotting. Suerias et al. [40] used an architecture inspired by sequence to sequence [41], in which they use an attention decoder rather than using the CRNN

outputs directly. Dutta et al. [12] compiled several recent advances in text recognition into a powerful architecture, reminiscent of modern networks for scene text recognition, as the ones presented recently by Baek et al. [3].

Handwriting text generation (HTG) is a relatively new field, brought forth by Graves [14], who introduced a method to synthesize online data based on a recurrent net. A modern extension of [14] was presented by Ji et al. [24], who followed the GAN paradigm [13] by adding a discriminator. DeepWriting [1] introduced better control over the style generation of [14] by disentangling it from the content.

Haines et al. [17] proposed a method to generate handwriting based on a specific author, but requires a time consuming character-level annotation process for each new data sample.

While all previous HTG methods demonstrate visually pleasing results, none were used to augment HTR training data, as opposed to the ones we discuss next.

Data augmentation using generative models. Generative models (and specifically GANs) are used to synthesize realistic data samples based on real examples. One possible use for these newly generated images is adding them to the original training set, essentially augmenting the set in a bootstrap manner. A recent example for this is the low-shot learning method by Wang et al. [42], who incorporate this process into the task loss in an end-to-end manner.

For the case at hand, we look at methods that use HTG or similar approaches to learn augmentation of the handwritten examples. One straightforward example of this is a method proposed by Bhunia et al. [5], who trains a GAN to warp the training set using a parametric function. Unlike ours, this approach cannot generate words outside a given lexicon, which is a crucial property as we show below (see Table 3). Krishanan el al. [28] proposed a method to harness synthetic data for word spotting, while not relying on a specific source of synthetic data (e.g. can use data made by our method).

Alonso et al. [2] presented a new HTG model reminiscent of the work in [42], which in turn inspired our approach. The network presented in [2] uses LSTM to embed the input word into a fixed length representation which can be fed into a BigGAN [8] architecture. As opposed to our approach, which allows for variable word and image length, this generator is only able to output images of a fixed width across all word lengths. Another large benefit of using a fully convolutional generator is removing the need to learn an embedding of the entire word using a recurrent network, we instead can learn the embeddings for each character directly without the need for character level annotation.

Another recent approach by Ingle et al. [22] uses an online generator similar to [14], followed by rendering. This approach is coupled with some synthetic generation of noise or other nuisance factors. Since this method relies on an online data generator, it cannot adapt to the versatility nor typical noise of an unseen *offline* dataset, which we claim is the common use case.

Classic augmentation is mentioned here mainly for completeness, including some methods that use less intricate ways to synthesize training examples, such as using handwriting fonts as proposed by [29]. Most of HTR methods mentioned above use some kind of randomized parametric spatial distortion to enlarge the visual variability of the data. Puigcerver [36] pushed this notion even further, and promoted that simpler one dimensional recurrent layers might be sufficient, if provided with data distortions.

3. Method

Our approach follows the GAN paradigm [13], where in addition to the discriminator $\mathcal{D}$, the resulting image is also evaluated by a text recognition network $\mathcal{R}$. While $\mathcal{D}$ promotes realistic looking handwriting styles, $\mathcal{R}$ encourages the result to be readable and true to the input text. This part of our architecture is similar to the one presented in [2], and is illustrated in the right side of Figure 2. This architecture minimizes a joint loss term ℓ from the two networks

$$\ell = \ell_{\mathcal{D}} + \boldsymbol{\lambda} \cdot \ell_{\mathcal{R}}, \tag{1}$$

where $\ell_{\mathcal{D}}$ and $\ell_{\mathcal{R}}$ are the loss terms of $\mathcal{D}$ and $\mathcal{R}$, respectively.

The main technical novelty of our method lies in the generator $\mathcal{G}$, as we describe next in Section 3.1. Other modifications made to the discriminator $\mathcal{D}$ and the recognizer $\mathcal{R}$ are covered in sections 3.2 and 3.3, respectively. We conclude by covering some optimization considerations on the parameter $\boldsymbol{\lambda}$ in Section 3.4.

3.1. Fully convolutional generator

The main observation guiding our design is that handwriting is a local process, i.e. when writing each letter is influenced only by its predecessor and successor. Evidence for this observation can be seen in previous works like [14], where the attention of the synthesizer is focused on the immediate neighbors of the current letter. This phenomenon is not trivial since the architecture in [14] uses a recurrent network, which we argue enforces no such constraint on the attention, but is rather 'free' to learn it.

Our generator is designed to mimic this process: rather than generating the image out of an entire word representation, as done in [2], each character is generated individually, using CNN's property of overlapping receptive fields to account for the influence of nearby letters. In other words, our generator can be seen as a concatenation of *identical* class

conditional generators [33] for which each class is a character. Each of these generators produces a patch containing its input character. Each convolutional-upsampling layer widens the receptive field, as well as the overlap between two neighboring characters. This overlap allows adjacent characters to interact, and creates a smooth transition.

The generation process is illustrated on the left side of Figure 2 for the word "meet". For each character, a filter $f_\star$ is selected from a filter-bank $\mathscr{F}$ that is as large as the alphabet, for example $\mathscr{F} = \{f_a, f_b, \ldots, f_z\}$ for lowercase English. Four such filters are concatenated in Figure 2 (f_e is used twice), and multiplied by a noise vector z, which controls the text style. As can be seen, the region generated from each character filter $f_\star$ is of the same size, and adjacent characters' receptive field overlap. This provides flexibility in the actual size and cursive type of the output handwriting character. For example, the letter "m" takes up most of the red patch, while the letters "e" and "t" take up a smaller portion of their designated patches, and the latter is the only non-cursive letter. Furthermore, learning the dependencies between adjacent characters allows the network to create different variations of the same character, depending on its neighboring characters. Such examples can be seen in Figure 1 and Figure 3.

The style of each image is controlled by a noise vector z given as input to the network. In order to generate the same style for the entire word or sentence, this noise vector is kept constant throughout the generation of all the characters in the input.

3.2. Style-promoting discriminator

In the GAN paradigm [13], the purpose of the discriminator $\mathcal{D}$ is to tell apart synthetic images generated by $\mathcal{G}$ from the real ones. In our proposed architecture, the role of $\mathcal{D}$ is also to discriminate between such images based on the handwriting output *style*.

The discriminator architecture has to account for the varying length of the generated image, and therefore is designed to be convolutional as well: The discriminator is essentially a concatenation of separate "real/fake" classifiers with overlapping receptive fields. Since we chose not to rely on character level annotations, we cannot use class supervision for each of these classifiers, as opposed to class conditional GANs such as [33, 8]. One benefit of this is that we can now use unlabeled images to train $\mathcal{D}$, even from other unseen data corpus. A pooling layer aggregates scores from all classifiers into the final discriminator output.

3.3. Localized text recognizer

While discriminator $\mathcal{D}$ promotes real-looking images, the recognizer $\mathcal{R}$ promotes readable text, in essence discriminating between gibberish and real text. Generated images are 'penalized' by comparing the recognized text in the output of $\mathcal{R}$ to the one that was given as input to $\mathcal{G}$. Following [2], $\mathcal{R}$ is trained only on real, labeled, handwritten samples.

Most recognition networks use a recurrent module, typically bidirectional LSTM [19], which reads the character in the current image patch by utilizing information from previous and subsequent image patches. As shown by Sabir el al. [37], the network learns an implicit language model which helps it identify the correct character even if it is not written clearly, by leveraging priors learned from other characters in the text. While this quality is usually desired in a handwriting recognition model, in our case it may lead the network to correctly read characters which were not written clearly by the generator. Therefore, we opted not to use the recurrent 'head' of the recognition network, which enables this quality, and keep only the convolutional backbone. See the supplementary material for a detailed analysis on this.

3.4. Optimization considerations

The generator network is optimized by the recognizer loss $\ell_\mathcal{R}$ and the adversarial loss $\ell_\mathcal{D}$. The gradients stemming from each of these loss terms can vary greatly in magnitude. Alonso et al. [2] proposed the following rule to balance the two loss terms

$$\nabla_I \mathcal{R} \leftarrow \alpha \left(\frac{\sigma(\nabla_I \mathcal{D})}{\sigma(\nabla_I \mathcal{R})} \cdot [\nabla_I \mathcal{R} - \mu(\nabla_I \mathcal{R})] + \mu(\nabla_I \mathcal{D}) \right), \tag{2}$$

where $\sigma(\cdot)$ and $\mu(\cdot)$ are respectively the empirical standard deviation and mean, $\nabla_I \mathcal{R}$ and $\nabla_I \mathcal{D}$ are respectively the gradients of $\ell_\mathcal{R}$ and $\ell_\mathcal{D}$ w.r.t. the image. The parameter α controls the relative importance of $\ell_\mathcal{R}$ compared to $\ell_\mathcal{D}$. In this paper, we chose to balance based only on the standard deviation of the losses and not the average

$$\nabla_I \mathcal{R} \leftarrow \alpha \left(\frac{\sigma(\nabla_I \mathcal{D})}{\sigma(\nabla_I \mathcal{R})} \cdot \nabla_I \mathcal{R} \right), \tag{3}$$

in order to avoid changing the sign of the gradient $\nabla_I \mathcal{R}$.

4. Results

4.1. Implementation details

Without loss of generality, the architecture is designed to generate and process images with fixed height of 32 pixels, in addition, the receptive field width of $\mathcal{G}$ is set to 16 pixels.

As mentioned in Section 3.1, the generator network $\mathcal{G}$ has a filter bank $\mathscr{F}$ as large as the alphabet, for example, $\mathscr{F} = \{f_a, f_b, \ldots, f_z\}$ for lowercase English. Each filter has a size of 32×8192. To generate one n-character word, we select and concatenate n of these filters (including repetitions, as with the letter 'e' in Figure 2), multiplying them with a 32 dimensional noise vector z_1, resulting in an $n \times 8192$ matrix. Next, the latter matrix is reshaped into a $512 \times 4 \times 4n$

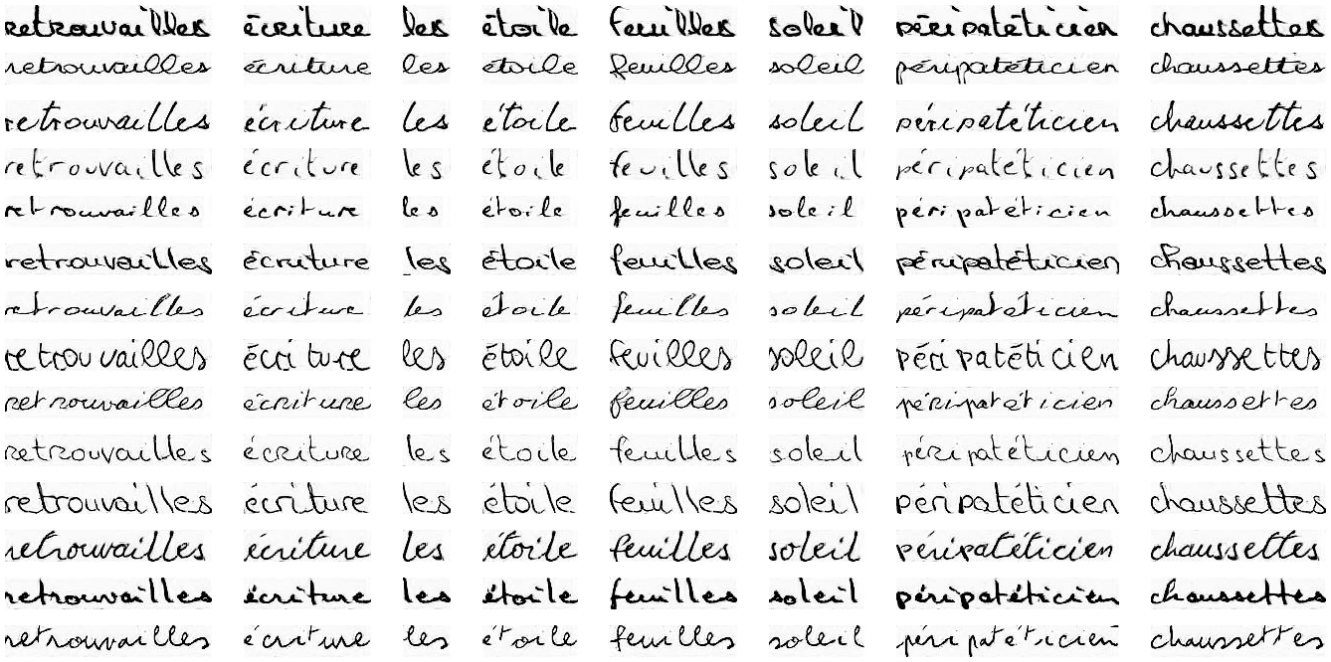


Figure 3: *Generating different styles.* Each row in the figure is generated by the same noise vector and results in the same handwriting style. The words generated in each column from left to right are: retrouvailles, écriture, les, étoile, feuilles, soleil, péripatéticien and chaussettes

Figure 4: Results of the work by Alonso et al. [2] (left column) vs our results (right column) on the words: olibrius, inventif, bonjour, ionique, malade, golf, ski, Dimanche, réparer, famille, gorille, certes, des, le.

tensor, i.e. at this point, each character has a spatial size of 4×4. The latter tensor is fed into three residual blocks which upsample the spatial resolution, create the aforementioned receptive field overlap, and lead to the final image size of $32 \times 16n$. Conditional Instance Normalization layers [11] are used to modulate the residual blocks using three additional 32 dimensional noise vectors, z_2, z_3 and z_4. Finally, a convolutional layer with a *tanh* activation is used to output the final image.

The discriminator network $\mathcal{D}$ is inspired by BigGAN [8]: 4 residual blocks followed by a linear layer with one output. To cope with varying width image generation, $\mathcal{D}$ is also fully convolutional, essentially working on horizontally overlapping image patches. The final prediction is the average of the patch predictions, which is fed into a GAN hinge-loss [30].

The recognition network $\mathcal{R}$ is inspired by CRNN [38]. The convolutional part of the network contains six convolutional layers and five pooling layers, all with ReLU activation. Finally, a linear layer is used to output class scores for each window, which is compared to the ground truth annotation using the *connectionist temporal classification* (CTC) loss [15].

Our experiments are run on a machine with one V100 GPU and 16GB of RAM. For more details on the architecture, the reader is referred to the supplemental materials.

4.2. Datasets and evaluation metrics

To evaluate our method, we use three standard benchmarks: RIMES[16], IAM [32], and CVL [27]. The RIMES dataset contains words from the French language, spanning about 60k images written by 1300 different authors. The IAM dataset contains about 100k images of words from the English language. The dataset is divided into words written by 657 different authors. The train, test and validation set contain words written by mutually exclusive authors. The CVL dataset consists of seven handwritten documents, out of which we use only the six that are English. These documents were written by about 310 participants, resulting in

about 83k word crops, divided into train and test sets.

All images were resized to a fixed height of 32 pixels while maintaining the aspect ratio of the original image. For the specific case of GAN training, and only when labels were used (supervised case), we additionally scaled the image horizontally to make each character approximately the same width as the synthetic ones, i.e. 16 pixels per character. This was done in order to challenge the discriminator by making real samples more similar to the synthesized ones.

We evaluate our method We evaluate our method using two common gold standard metrics. First, word error rate (WER) is the number of misread words out of the number of words in the test set. Second, normalized edit-distance (NED) is measured by the edit-distance between the predicted and true word normalized by the true word length. Whenever possible, we repeat the training session five times and report the average and standard deviation thereof.

4.3. Comparison to Alonso el al. [2]

Since no implementation was provided, we focus on qualitative comparison to [2] using images and metrics presented therein. Figure 4 contains results shown in [2] alongside results of our method on the same words. As can be seen in the figure, our network produces images that are much clearer, especially for shorter words. More generally, our results contain fewer artifacts, for example, the letter 'm' in the fifth row, the redundant letter 'i' in the sixth row and the missing 's' in the row before last.

Table 4 compares the two methods using standard metrics for GAN performance evaluation, namely *Fréchet Inception Distance* (FID) [18] and *geometric-score* (GS) [25]. Using a similar setting[1] to the ones described in [2], our method shows slightly better performance on both metrics. Note, however, that since we do not have access to the data from [2], both metrics for that method are copied from the paper, and hence cannot be used to directly compare to our results.

4.4. Generating different styles

We are able to generate different handwriting styles by changing the noise vector z that is fed into ScrabbleGAN. Figure 3 depicts examples of selected words generated in different handwriting styles. Each row in the figure represent a different style, while each column contains a different word to synthesize. As can be seen in the figure, our network is able to generate both cursive and non-cursive text, with either a bold or thin pen stroke. This image provides a good example of character interaction: while all repetitions of a character start with identical filters f_i, each final instantiation might be different depending on the adjacent characters.

[1] We ran this experiment once, as opposed to [2] who presented the best result over several runs

	FID	GS
Alonso el al. [2]	23.94	8.58×10^{-4}
ScrabbleGAN	**23.78**	$\mathbf{7.60 \times 10^{-4}}$

Table 1: Comparison of our method to Alonso et al.[2] using Fréchet Inception Distance and geometric-score metrics. Lower values are better.

Figure 5 shows interpolations between two different styles on the IAM dataset. In each column we chose two random noise vectors for the first and last row, and interpolated between them linearly to generate the noise vectors for the images in between. The size of each letter, the width of the pen strokes and the connections between the letters change gradually between the two styles. The gray background around the letters is a property of the original IAM dataset and can be found in most of the images in the dataset. As a result, the generator also learns to generate variations of the background.

4.5. Boosting HTR performance

Our primary motivation to generate handwriting images is to improve the performance of an HTR framework compared to the "vanilla" supervised setting. For all experiments in this section, we use the code provided by [3] as our HTR framework, as it contains all the improvements presented in [12] (for which no implementation was provided), as well as some other recent advances that achieve state of the art performance on the scene text recognition problem for printed text. We show that training the best architecture in [3] on the handwritten data yields performance close to state of the art on HTR, which should be challenging to improve upon. Specifically, our chosen HTR architecture is composed of a thin plate spline (TPS) transformation model, a ResNet backbone for extracting the visual features, a bi-directional LSTM module for sequence modeling, and an attention layer for the prediction. In all the experiments, we used the validation set to choose the best performing model, and report the performance thereof on its associated test set.

Train set augmentation is arguably the most straightforward application of a generative model in this setting: by simply appending generated images to the train set, we strive to improve HTR performance in a bootstrap manner. Table 2 shows WER and NED of the HTR network when trained on various training data agumentations on the training data, for both RIMES and IAM datasets, where each row adds versatility to the process w.r.t. its predecessor. For each dataset, the first row shows results when using the original training data, which is the baseline for comparison. Next, the second row shows performance when the data

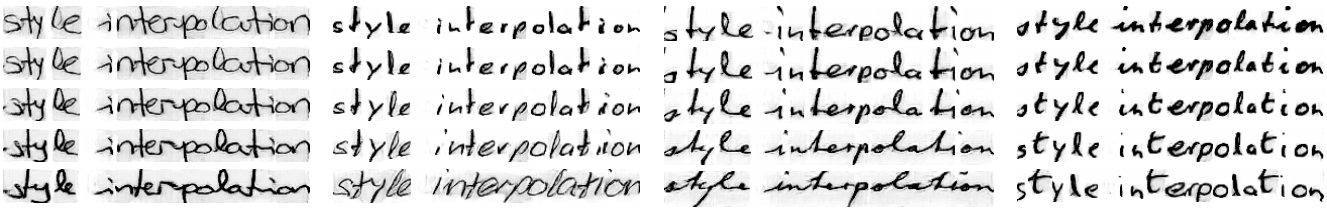


Figure 5: *Style interpolation.* Each column contains an interpolation between two different styles of handwriting generated by ScrabbleGAN. Note that the GAN captures the background noise typical to the IAM dataset [32].

is augmented with a random affine transformations. The third row shows results using the original training data and an additional 100k synthetic handwriting image generated by ScrabbleGAN. The last row further fine-tunes the latter model using the original training data. As can be seen in the table, using the ScrabbleGAN generated samples during training leads to a significant improvement in performance compared to using only off-the-shelf affine augmentations.

Set	Aug	GAN	Refine	WER[%]	NED[%]
RIMES	×	×	-	12.29 ± 0.15	3.91 ± 0.08
	✓	×	-	12.24 ± 0.2	3.81 ± 0.08
	✓	100k	×	11.68 ± 0.29	3.74 ± 0.10
	✓	100k	✓	11.32 ± 0.31	3.57 ± 0.13
IAM	×	×	-	25.10 ± 0.49	13.82 ± 0.35
	✓	×	-	24.73 ± 0.53	13.98 ± 0.93
	✓	100k	×	23.98 ± 0.4	13.57 ± 0.24
	✓	100k	✓	23.61 ± 0.36	13.42 ± 0.27

Table 2: *HTR experiments on RIMES and IAM*. For each dataset we report four results with gradually increasing versatility to the dataset w.r.t. its predecessor. The second column ('Aug') indicates usage of random affine augmentation in train time. The third column ('GAN') indicates whether synthetic images were added to the original train set, and how many. The fourth column ('Refine') indicates whether another pass of fine tuning was performed using the original data. See text for more details.

Domain adaptation, sometimes called transductive transfer learning, is the process of applying a model on data from a different distribution than the one it was trained on. We test this task by transferring from IAM to CVL as they both use the same alphabet and are somewhat visually similar. One naive solution for this is training a model on the IAM dataset, and testing its performance on the CVL test set. This will be our baseline for comparison. Since ScrabbleGAN can be trained on unlabeled data, it can adapt to the style of CVL images without using the ground truth. We synthesize data according three different flavors: using either CVL style, CVL lexicon, or both (as opposed to IAM).

Train data	Style	Lex.	WER[%]	NED[%]
IAM (naive)	N/A	IAM	39.95 ± 0.91	19.29 ± 0.95
IAM+100K	CVL	IAM	40.24 ± 0.51	19.49 ± 0.76
IAM+100K	IAM	CVL	35.98 ± 0.38	17.27 ± 0.23
IAM+100K	CVL	CVL	29.75 ± 0.67	$\mathbf{14.52 \pm 0.51}$
CVL (oracle)	N/A	CVL	$\mathbf{22.90 \pm 0.07}$	15.62 ± 0.15

Table 3: *Domain adaptation results* from the IAM dataset to the CVL dataset. First row is naive approach of using a net trained on IAM. Next three rows show the effect of 100k synthetic images having either CVL style, CVL lexicon or both. The bottom row shows the oracle performance of supervised training on the CVL train set, just for reference. No CVL labels were used to train HTR, except for the oracle.

Data generated from each of these three flavors is appended to the IAM training set, as we find this helps stabilize HTR training. Finally, we set a "regular" supervised training session of CVL train set, to be used as an oracle, i.e. to get a sense of how far we are from using the train labels.

Table 3 summarizes performance over the CVL test set of all the aforementioned configurations, ranging from the naive case, through the flavors of using data from ScrabbleGAN, to the oracle. First, we wish to emphasize the 17% WER gap between the naive approach and the oracle, showing how hard it is for the selected HTR to generalize in this case. Second, we observe that synthesizing images with CVL style and IAM lexicon (second row) does not alter the results compared to the naive approach. On the other hand, synthesizing images with IAM style and CVL lexicon (third row) boosts WER performance by about 5%. Finally, synthesizing images with both CVL style and lexicon (fourth row) yields another 5% boost in WER, with NED score that is better than the oracle.

4.6. Gardient balancing ablation study

Several design considerations regarding parameter selection were made during the conception of ScrabbleGAN. We focus on two main factors: First, the effect of gradient balancing (GB) presented below, and second, the surprising ef-

GB Type	α	WER[%]	NED[%]
No GB	-	12.64 ± 0.20	4.18 ± 0.11
[2]	1	12.83 ± 0.28	4.21 ± 0.06
Ours	0.1	12.28 ± 0.49	3.95 ± 0.26
Ours	1	$\mathbf{11.68 \pm 0.29}$	$\mathbf{3.74 \pm 0.10}$
Ours	10	12.03 ± 0.27	3.80 ± 0.04

Table 4: *GB ablation study*, comparing HTR performance trained on different synthetic datasets. Each such set was generated by a GAN with different GB scheme. See text for details.

fect of the architecture of the recognizer $\mathcal{R}$ which we leave to the supplementary material.

Table 4 compares HTR results on the RIMES dataset using three different variations of gradient balancing during training: First, we show results when no gradient balancing is used whatsoever. Second, we apply the gradient balancing scheme suggested in [2], which is shown in Eq. (2). Finally, we show how our modified version performs for different values of the parameter α, as described in Eq. (3). For all the above options we repeat the experiment shown in the third row of Table 2, and report WER and NED scores. Clearly, the best results are achieved using samples synthesized from a GAN trained using our gradient balancing approach with $\alpha = 1$.

Figure 6 further illustrates the importance of balancing between $\ell_{\mathcal{R}}$ and $\ell_{\mathcal{D}}$ and the effect of the parameter α. Each column in the figure represents a different value starting from training only with $\ell_{\mathcal{R}}$ on the left, to training only with $\ell_{\mathcal{D}}$ on the right. The same input text, "ScrabbleGAN", is used in all of the images and the same noise vector is used to generate each row. As expected, using only the recognizer loss results in images which look noisy and do not contain any readable text. On the other hand, using only the adversarial loss results in real-looking handwriting images, but do not contain the desired text but rather gibberish. A closer look at this column reveals that manipulating the value of z changes the letter itself, rather than only the style. From left to right, the three middle columns contain images generated by a GAN trained with α values of 10, 1, and 0.1. The higher the value of α is, the higher the weight of the $\ell_{\mathcal{R}}$ is. The results using $\alpha = 10$ are all readable, but contain much less variability in style. Conversely, using $\alpha = 0.1$ yields larger variability in style at the expense of the text readability, as some of the letters become unrecognizable. The images depicted in Figure 6 provide another explanation for the quantitative results shown in Table 4. Training an HTR network with images generated by a GAN trained with larger α deteriorates the results on diverse styles, while training with images generated by a GAN trained with a smaller α value might lead to recognition mistakes caused by training on unclear text images.

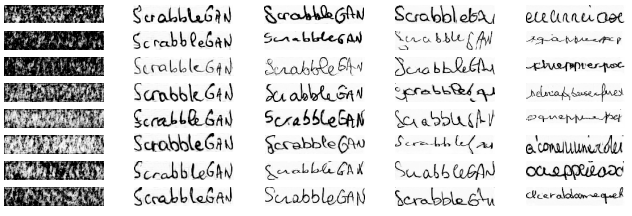


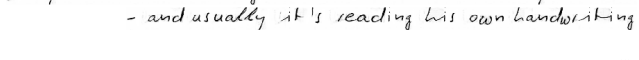
Figure 6: Comparison of different balancing levels between $\ell_{\mathcal{D}}$ and $\ell_{\mathcal{R}}$, the discriminator and recognizer loss terms, respectively. Setting α's value to ∞ or 0 means training only with $\mathcal{R}$ or $\mathcal{D}$, respectively. All examples are generation of the word "ScrabbleGAN", where each row was generated with the same noise vector z.

5. Conclusion and Future Work

We have presented a new architecture to generate offline handwritten text images, which operates under the assumption that writing characters is a local task. Our generator architecture draws inspiration from the game "Scrabble". Similarly to the game, each word is constructed by assembling the images generated by its characters. The generated images are versatile in both stroke widths and general style. Furthermore, the overlap between the receptive fields of the different characters in the text enables the generation of cursive as well as non-cursive handwriting. We showed that the large variability of words and styles generated, can be used to boost performance of a given HTR by enriching the training set. Moreover, our approach allows the introduction of an unlabeled corpus, adapting to the style of the text therein. We show that the ability to generate words from a new lexicon is beneficial when coupled with the new style.

An interesting avenue for future research is to use a generative representation learning framework such as VAE [26] or BiGAN [9, 10], which are more suitable for few shot learning cases like author adaptation. Additionally, disentanglement approaches may allow finer control of text style, such as cursive-ness or pen width.

In the future, we additionally plan to address the fact that generated characters have the same receptive field width. This is, of course, not the case for most scripts, as 'i' is usually narrower than 'w', for example. One possible remedy for this is having a different width for each character filter depending on its average width in the dataset. Another option is to apply STN [23] as one of the layers of $\mathcal{G}$, in order to generate a similar effect.



References

[1] Emre Aksan, Fabrizio Pece, and Otmar Hilliges. Deepwriting: Making digital ink editable via deep generative modeling. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2018. 3

[2] Eloi Alonso, Bastien Moysset, and Ronaldo Messina. Adversarial generation of handwritten text images conditioned on sequences. *arXiv preprint arXiv:1903.00277*, 2019. 3, 4, 5, 6, 8, 12

[3] 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. 3, 6, 11, 12

[4] BBC, Is stenography a dying art?, https://www.bbc.com/news/magazine-13035979, 2019-11-01. 1

[5] Ayan Kumar Bhunia, Abhirup Das, Perla Sai Raj Kishore, Shuvozit Ghose, and Partha Pratim Roy. Handwriting recognition in low-resource scripts using adversarial learning. *arXiv preprint arXiv:1811.01396*, 2018. 3

[6] 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. 12

[7] Andy Brock and Alex Andonian, Biggan-pytorch, https://github.com/ajbrock/BigGAN-PyTorch, 2019-11-01. 12

[8] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 3, 4, 5, 12

[9] Jeff Donahue, Philipp Krhenbhl, and Trevor Darrell. Adversarial feature learning. In *5th International Conference on Learning Representations, ICLR 2017*, 2017. 8

[10] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. In *5th International Conference on Learning Representations, ICLR 2017*, 2017. 8

[11] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. 2017. 5, 11

[12] Kartik Dutta, Praveen Krishnan, Minesh Mathew, and CV Jawahar. Improving cnn-rnn hybrid networks for handwriting recognition. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 80–85. IEEE, 2018. 3, 6

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 3, 4

[14] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. 3

[15] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006. 5

[16] Emmanuèle Grosicki and Haikal El Abed. Icdar 2009 handwriting recognition competition. In *2009 10th International Conference on Document Analysis and Recognition*, pages 1398–1402. IEEE, 2009. 5

[17] Tom S.F. Haines, Oisin Mac Aodha, and Gabriel J. Brostow. My Text in Your Handwriting. In *Transactions on Graphics*, 2016. 3

[18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017. 6

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 4

[20] Amazon Inc., Amazon textract, https://aws.amazon.com/textract, 2019-11-01. 1

[21] Google Inc., Detect text in images, https://cloud.google.com/vision/docs/ocr, 2019-11-01. 1

[22] R. Reeve Ingle, Yasuhisa Fujii, Thomas Deselaers, Jonathan Baccash, and Ashok C. Popat. A scalable handwritten text recognition system. *ArXiv*, abs/1904.09150, 2019. 3

[23] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015. 8

[24] Bo Ji and Tianyi Chen. Generative adversarial network for handwritten text. *arXiv preprint arXiv:1907.11845*, 2019. 3

[25] Valentin Khrulkov and Ivan Oseledets. Geometry score: A method for comparing generative adversarial networks. *arXiv preprint arXiv:1802.02664*, 2018. 6

[26] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 8

[27] Florian Kleber, Stefan Fiel, Markus Diem, and Robert Sablatnig. Cvl-database: An off-line database for writer retrieval, writer identification and word spotting. In *2013 12th International Conference on Document Analysis and Recognition*, pages 560–564. IEEE, 2013. 5

[28] Praveen Krishnan, Kartik Dutta, and CV Jawahar. Word spotting and recognition using deep embedding. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 1–6. IEEE, 2018. 3

[29] Praveen Krishnan and C. V. Jawahar. Generating synthetic data for text recognition, 2016. 3

[30] Jae Hyun Lim and Jong Chul Ye. Geometric gan. *arXiv preprint arXiv:1705.02894*, 2017. 5

[31] Shu Liyang, Crnn-pytorch, https://github.com/-Holmeyoung/crnn-pytorch, 2019-11-01. 11, 12

[32] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002. 5, 7

[33] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 4

[34] Réjean Plamondon and Sargur N Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):63–84, 2000. 2

[35] Arik Poznanski and Lior Wolf. Cnn-n-gram for handwriting word recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2305–2314, 2016. 2

[36] Joan Puigcerver. Are multidimensional recurrent layers really necessary for handwritten text recognition? In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 67–72. IEEE, 2017. 3

[37] Ekraam Sabir, Stephen Rawls, and Prem Natarajan. Implicit language model in lstm for ocr. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 7, pages 27–31. IEEE, 2017. 4

[38] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016. 2, 5

[39] Sebastian Sudholt and Gernot A Fink. Phocnet: A deep convolutional neural network for word spotting in handwritten documents. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 277–282. IEEE, 2016. 2

[40] Jorge Sueiras, Victoria Ruiz, Angel Sanchez, and Jose F Velez. Offline continuous handwriting recognition using sequence to sequence neural networks. *Neurocomputing*, 289:119–128, 2018. 2

[41] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014. 2

[42] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7278–7286, 2018. 3

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

A. Visual Results

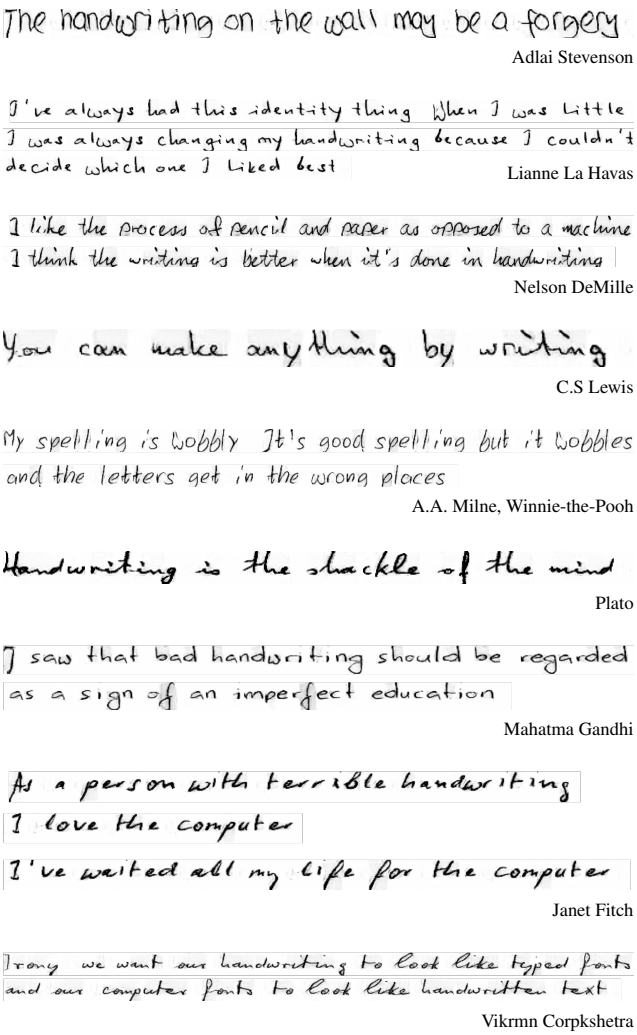


Figure 7: Quotes about handwriting. All these examples were originally one single image, and some where split into several lines to fit one column.

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 7. Each quote was originally one single image, and was split into several lines to fit one column.

B. Ablation Study

Ablation results. In Table 5 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 [31]	**11.68 ± 0.29**	**3.74 ± 0.10**
CNN [31] + 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 [31] w/o CBN [11]	12.46 ± 0.30	4.01± 0.09

Table 5: *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 5 present three alternatives from the code provided by [3]. 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 [31], which contains a simple feed forward backbone of seven convolutional layers with a bidirectional LSTM on top. We tested this architecture with- and with-

out the LSTM module, and respectively present their performance in rows 2 and 1 of Table 5. Indeed, this simpler network helped the GAN generate the best images to be used for HTR training. Alonso el al. [2] used gated CRNN as their recognizer $\mathcal{R}$, originally presented in [6]. Since this is very similar to the CRNN presented in [3], and no implementation of [6] was provided, we chose not to include an evaluation of this specific architecture.

GAN noise input selection. As we describe in Section C 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 5, doing so adds about 1% to the WER score. Therefore, we opted to use CBN layers in the generator.

C. 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 6: 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 7: 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 [7]. 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 6 and 7 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 [8] 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 in Section 4.1 in the paper.

Recognizer. For $\mathcal{R}$ we based our implementation on the RCNN implementation by [31]. In light of the ablation presented in section B, we decided to remove the Bi-LSTM network.