JokerGAN: Memory-Efficient Model for Handwritten Text Generation with Text Line Awareness

Jan Zdenek
jan@nlab.ci.i.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

Hideki Nakayama
nakayama@ci.i.u-tokyo.ac.jp
The University of Tokyo
Tokyo, Japan

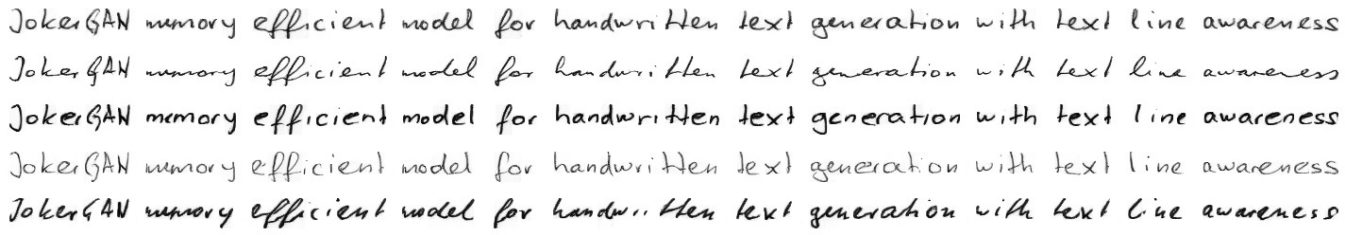

Figure 1: The text in this image was generated by our proposed method.

ABSTRACT

Collecting labeled data for training of models for image recognition problems, including handwritten text recognition (HTR), is a tedious and expensive task. Recent work on handwritten text generation shows that generative models can be used as a data augmentation method to improve the performance of HTR systems.

In this paper, we propose a new method for handwritten text generation that uses generative adversarial networks with multi-class conditional batch normalization, which enables us to use character sequences with variable lengths as conditional input for the generator. Compared to existing methods, our method has significantly lower memory requirements which are almost constant regardless of the size of the character set. This allows us to train a generative model for languages with a large number of characters, such as Japanese. We also introduce an additional condition that makes the generator aware whether there are characters extending below the baseline or above the mean line in the generated sequence, which helps generate handwritten text with well-aligned characters in the text line.

Experiments on handwritten text datasets show that our proposed model can be used to boost the performance of HTR, particularly when we only have access to partially annotated data and train our generative model in semi-supervised fashion. The results also show that our model outperforms the current state-of-the-art for handwritten text generation. In addition, we perform a human evaluation study that indicates that the proposed method generates handwritten text images that look more realistic and natural.

ACMMM '21, October 20–24, 2021, Chengdu, China

ACM ISBN 978-1-4503-XXXX-X/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**; *Computer graphics.*

KEYWORDS

generative adversarial networks, image generation, handwritten text, data augmentation

ACM Reference Format:
Jan Zdenek and Hideki Nakayama. 2021. JokerGAN: Memory-Efficient Model for Handwritten Text Generation with Text Line Awareness. In *ACMMM '21: ACM International Conference on Multimedia, October 20–24, 2021, Chengdu, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Handwriting has been used since the distant past as a means of documentation and communication. In the digital era, a lot of handwritten text has been replaced by text in digital form; however, handwriting is still used in many different situations and places where handwriting is more convenient or using digital devices is difficult. One of the prominent places where a lot of data is still recorded using handwritten documents are medical institutions. Recently, there is an increasing demand for technology that can convert handwritten data into digital form for various purposes such as storing and sharing data more securely [4], creating searchable databases [1], and analyzing data with modern technologies [9].

State-of-the-art systems for optical character recognition (OCR) can recognize text in printed form with excellent precision, but recognition of handwritten text is still quite a challenging problem. Recent methods for handwritten text recognition (HTR) use deep neural networks (DNNs) since DNNs have demonstrated exceptional performance in many computer vision tasks. Modern DNNs benefit from having access to a large amount of labeled data, and increasing the size of the training set tends to improve the accuracy. However, collecting labeled data for training is a tedious and

costly process. In addition, DNNs are sensitive to differences in image characteristics that can be caused by using different types of pens and paper, different lighting conditions, and different scanning techniques and post-processing. As a result, HTR systems trained on existing handwritten text datasets might not yield good performance on images of handwritten text with different properties, and creating new labeled datasets can lead to better results.

Recently, several works have made an effort to improve the performance of handwritten text recognition systems by exploiting the progress in image generation [7, 14, 30, 31], and use generative models to augment training datasets by creating new training samples. Alonso et al. [3] were the first to use handwritten text generation as a data augmentation method to increase HTR performance. However, their method can only generate images of a fixed size, which is not suitable for generating handwritten words because words greatly vary in length. Another work [12] proposed an improvement that allows generation of images of variable sizes, which results in a significantly better visual quality of the results. However, the method requires a bank of filters for each character in the dataset. This makes the size of the model nearly linearly dependent on the size of the character set and makes training on datasets of languages with large character sets such as Japanese infeasible.

In this work, we propose a novel method for handwritten text generation to produce more realistic images of handwritten text. Our main motivation is to improve HTR performance by data augmentation using trained generative models. However, we believe that automatic generation of handwritten text can also be used for various other purposes such as generating realistic-looking handwritten text in games or virtual reality environments.

The contributions of our paper are as follows:

- We propose a novel method for handwritten text generation which can generate images of text with variable lengths and use arbitrary character sequences as conditions. It outperforms current state-of-the-art in several experiments while having significantly lower memory requirements achieved by introducing multi-class conditional batch normalization.
- Thanks to the lower memory requirements, our method can be used to train generative models using datasets with large character sets, such as Japanese text, which contains thousands of different characters.
- We introduce a novel conditional input that specifies vertical properties of characters in the generated word, which improves alignment of generated characters in the text line.

Our proposed method is inspired by ScrabbleGAN [12], and so is the name of it. ScrabbleGAN owes its name to an analogy between its generation process and the way that words are put together in the game of Scrabble, i.e. concatenating tiles with different letters to form a word. In our proposed method, we chain a single base filter into the desired length based on the target word length. Our single base filter can turn into any letter in the character set based on the conditional input into the generator, which resembles the blank tile in Scrabble or a joker in cards. Therefore, we call our proposed method JokerGAN.

2 RELATED WORK

2.1 Handwritten Text Recognition

Handwritten text recognition (HTR) is a special case of optical character recognition that focuses on handwritten text. It is an extensively studied research field that has attracted a lot of interest over many years [32]. HTR methods can be divided into two types: online and offline. Online methods [6, 19] use sequential data of pen location as the text is being written. This information about how the text was written provides more clues than an image and it can help identify characters based on the movement of pen. Offline methods [11, 33, 37] only use images of handwritten text, so they do not have access to the information about how the text was written. However, since offline methods only use images of text, it is easier to collect a lot of such data for training. Furthermore, offline methods can be used in a broader variety of situations where online data cannot be obtained.

Recent offline HTR methods exploit the progress in deep neural networks. Suerias et al. [37] proposed a method that is inspired by the sequence-to-sequence model [38], and uses an attention decoder for prediction. Another recent work [11] achieved improvement in HTR performance by combining several popular techniques for text recognition and introduced a model that shares a lot of similarities with recent state-of-the-art models for scene text recognition, which suggests that a lot of approaches for scene text recognition can be used for HTR, too.

2.2 Handwritten Text Image Generation

Success of deep learning methods in the past decade has seen deep neural networks being used for many new problems and applications, including handwritten text generation. The first work that applied deep learning to handwritten text generation proposed a method for synthesis of online handwritten trajectories using a recurrent neural network [15]. Ji et al. [20] improved the method from [15] by adding a discriminator network and using adversarial training introduced by GANs [14]. Further improvement was achieved in [2] by disentangling the style and content of handwritten text images to add control over the style of generated handwritten text.

Collecting large amount of online handwriting data is challenging because special equipment is required to record the sequential data of handwriting trajectories. Furthermore, it is difficult to learn long-range dependencies using recurrent-based methods, and their training requires a lot of time. On the other hand, obtaining images of handwritten text is much easier and recent advancements in image generation [7, 14, 30, 31] further encourage to research handwritten text generation using offline handwriting data.

Both GANs [14] and variational auto-encoders [23] were shown to be able to generate images of realistic handwritten digits. Later, Mirza et al. proposed conditional GANs (cGANs) [30] that were able to generate images of single handwritten digits conditioned on their class label.

Research on generation of handwritten text images can be divided into two categories: 1) generation of handwritten text with styles based on latent vectors sampled from a given distribution [3, 12], and 2) generation of handwritten text with style conditioned by reference images [10, 13, 21]. Some methods such as [13] also

use additional information about writer identities for training. In our work, we focus on the first category.

Alonso et al. [3] were the first to apply recent progress in image generation to propose a method that could generate images of handwritten words. Furthermore, they demonstrated that such generated images can be used to augment datasets of handwritten text and improve performance in handwritten text recognition. The architecture of their proposed model was based on BigGAN [7], and they used LSTM [18] to embed the target word into a fixed length vector which can be used as a condition for the generator. As a result of the design, the generator only produces fixed-size images, which means that the width of the generated handwritten words is the same regardless of the number of characters in the word, which can cause distortion. ScrabbleGAN [12] was inspired by [3] and it uses a similar architecture based on BigGAN. Unlike [3], ScrabbleGAN can generate images of various sizes depending on the length of the target word. Instead of LSTM word embeddings, it uses a bank of base filters for each letter in the alphabet. The generator input has a variable size and consists of k base filters that correspond to k letters in the target word. The base filters are concatenated and passed to the generator. The main disadvantage of ScrabbleGAN is that the memory requirements for the bank of base filters grow significantly as the size of the character set increases. In case of the Latin alphabet, the size of the character set is small and does not pose problems, but the memory requirements for training with a dataset of Chinese or Japanese text are too large to allow conventional training on regular GPUs.

In contrast with ScrabbleGAN, our proposed method can generate handwritten text of variable size without using a bank of base filters. As a result, our model requires significantly less memory and it can scale to data with a large number of characters in the character set. In addition, we introduce a new type of conditioning that improves the alignment of characters in generated images.

2.3 Image Generation of Chinese Characters

Chang et al. [8] proposed a generative model based on CycleGAN [40] to synthesize images of handwritten Chinese characters from Chinese characters in printed form. Several more works followed and proposed methods to generate Chinese handwriting [39] and fonts for Chinese characters [26] from existing images using image translation and style transfer techniques. To the best of our knowledge, our work is the first to attempt generation of handwritten Chinese characters from scratch.

3 JOKERGAN

Our method is based on GANs for image generation and is inspired by [12]. Our objective is to train a generator $\mathcal{G}$ that can produce realistic images of handwritten text conditioned by character sequences of arbitrary length $\mathbf{y} = (y_1, \ldots, y_k)$, where k is the length of the character sequence. Besides a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$, which are essential to train a GAN model, we also use a network for text recognition $\mathcal{R}$. During training, images generated by $\mathcal{G}$ are passed to both $\mathcal{D}$ and $\mathcal{R}$. The purpose of $\mathcal{D}$ is to predict whether the image is real or generated, which helps $\mathcal{G}$ to learn to generate realistic looking handwriting images by using an adversarial loss. The recognition network $\mathcal{R}$ tries to recognize the text in the image and encourages $\mathcal{G}$ to generate images that are readable and match the input condition using a loss function for text recognition.

In Section 3.1, we introduce the architecture of each module in our model, and explain the training process.

In our proposed method, we modify the generator from [12] and replace the filter bank by a single base filter for all characters. In addition, we introduce multi-class conditional batch normalization. It replaces the functionality of the filter bank and allows the network to generate images of character sequences that match the input condition $\mathbf{y}$ while significantly reducing the memory requirements of the network. We describe it in detail in Section 3.2. Furthermore, we also introduce an additional condition based on the vertical aspects of the text to make the generator $\mathcal{G}$ aware of the global baseline and mean line of the text, which is covered in Section 3.3.

3.1 Model Architecture

Discriminator. The discriminator $\mathcal{D}$ learns a binary classification problem of predicting whether an image is real or generated by $\mathcal{G}$. A pooling layer is used to aggregate the outputs of the last convolutional layer, which allows image inputs with variable sizes. The aggregated features are used to make the final binary prediction.

Given a dataset with pairs of images $\mathbf{x}$ and their labels $\mathbf{y}$, latent vectors $\mathbf{z}$ sampled from a normal distribution $\mathcal{N}(0, 1)$, and text line conditions $\mathbf{t}$ (explained in Section 3.3), $\mathcal{D}$ is optimized by minimizing hinge adversarial loss [25] defined as follows:

$$\begin{aligned}\mathcal{L}_{adv}^{\mathcal{D}} =& \mathbb{E}_{\mathbf{x}\sim p_{data}}[\min(0, -1 + \mathcal{D}(\mathbf{x})] \\ & - \mathbb{E}_{\mathbf{z}\sim p_z, \mathbf{y}\sim p_{data}}[\min(0, -1 - \mathcal{D}(\mathcal{G}(\mathbf{y}, \mathbf{z}, \mathbf{t}))].\end{aligned} \tag{1}$$

Text Recognizer. The text recognizer $\mathcal{R}$ predicts the text in the input images. Most recent methods for text recognition [27, 28, 35] use bidirectional recurrent neural networks [34], which allows them to use information from the entire character sequence to predict individual characters by learning an implicit language model. This can allow the network to read some characters correctly despite being illegible. That is a desired property for text recognition, but we want $\mathcal{R}$ to promote generation of legible characters, so this property is undesirable in our case. Therefore, following [12], we do not use recurrent layers in our $\mathcal{R}$.

The text recognizer $\mathcal{R}$ is trained only on real labeled images of handwritten words. We optimize $\mathcal{R}$ by minimizing the connectionist temporal classification (CTC) loss [16] using pairs of images $\mathbf{x}$ and their corresponding ground-truth labels $\mathbf{y}$ from the labeled dataset.

$$\mathcal{L}_{CTC}^{\mathcal{R}} = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{data}}[-\mathbf{y} \cdot \log \mathcal{R}(x)]. \tag{2}$$

Training $\mathcal{R}$ to correctly recognize handwritten text allows us to use it to provide guidance to training of $\mathcal{G}$.

Generator. We want to generate images of handwritten text conditioned by character sequences with arbitrary lengths, so the generator has to be able to generate images of varying sizes. In handwriting, all characters are typically written next to each other and each character can affect the shape of its neighboring characters or it can even be connected to them. The design of our generator mimics this process by the following strategy.

(1) **Text Base Map Formation**. Let f be a base filter of a generator network. We implement f as a linear neural network layer. Given a character sequence $\mathbf{y} = (y_1, \ldots, y_k)$ of size k

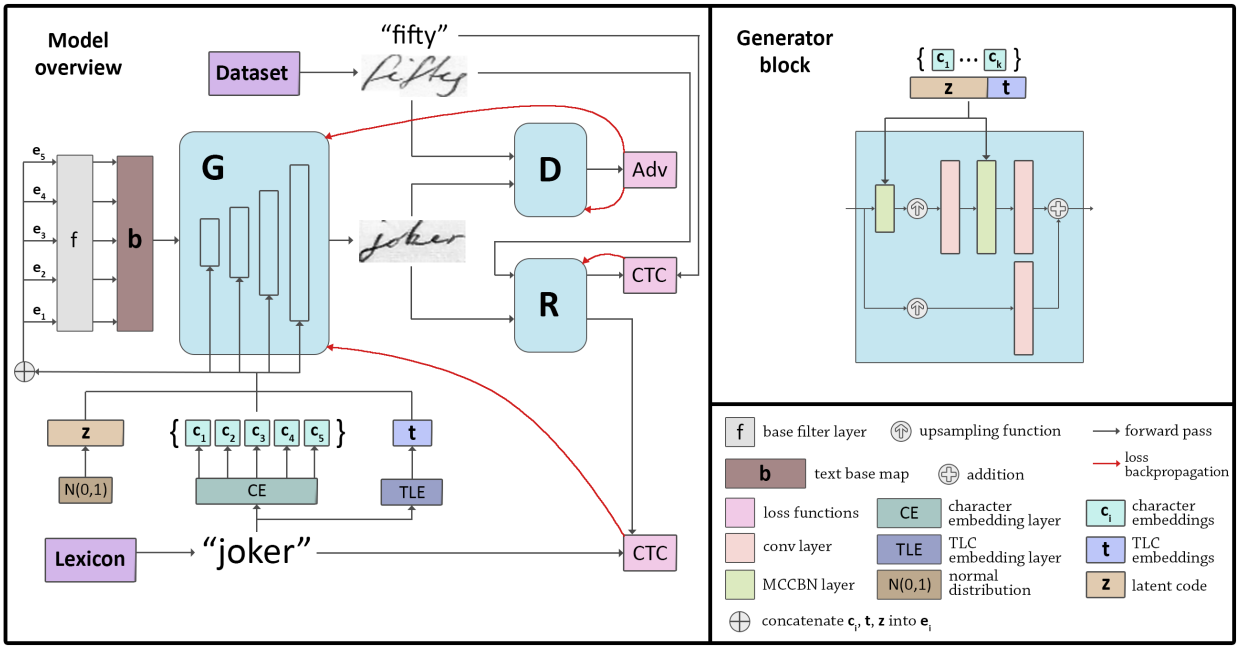

Figure 2: An overview of JokerGAN. During training, the generator is updated using both the adversarial loss and the CTC loss for text recognition. The discriminator learns to discern real images of handwriting and generated images without using any information about classes. The text recognition module is optimized only using real images.

that we want to generate, and its embedding $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_k)$, we concatenate $\mathbf{c}$ with a latent code $\mathbf{z}$ sampled from the normal distribution, and text line condition embeddings $\mathbf{t}$, described in 3.3, to form embeddings $\mathbf{e} = (\mathbf{c}_1 \oplus \mathbf{z} \oplus \mathbf{t}, \ldots, \mathbf{c}_k \oplus \mathbf{z} \oplus \mathbf{t}) = (\mathbf{e}_1, \ldots, \mathbf{e}_k)$, where $\oplus$ denotes concatenation. Each element $\mathbf{e_i}$ in $\mathbf{e}$ represents a single character in the character sequence $\mathbf{y} = (y_1, \ldots, y_k)$ used as condition. We pass $\mathbf{e}$ to the base filter f, and concatenate the outputs from f along the horizontal axis as if putting tiles next to each other to create a text base map $\mathbf{b}$.

The major benefit and main difference from [12] is that our model uses only one base filter for all characters in the character set. The generator in [12] requires a bank of base filters $\mathcal{F}$ which is as large as the character set. In case of the English alphabet, the filter bank $\mathcal{F} = \{f_a, ..., f_z, f_A, ..., f_Z\}$ consists of 52 base filters. The base filters have a lot of parameters and have a significant memory footprint, so using a large bank of base filters is very inefficient.

(2) **Conditional Generation**. A fully convolutional network is used to upsample the text base map and generate characters according to a given character sequence $\mathbf{y} = (y_1, \ldots, y_k)$. We introduce multi-class conditional batch normalization (MCCBN) which is our extension of conditional batch normalization (CBN) [31] that is widely used in conditional image generation. Unlike CBN, which conditions the whole image using one class, MCCBN lets us condition the image generation process on a sequence of multiple classes, which is the character sequence $\mathbf{y}$ embedded into $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_k)$ in our case. Besides class conditions, MCCBN also allows us to inject a latent vector $\mathbf{z}$, which we sample from a normal distribution $\mathcal{N}(0, 1)$, to generate text in different styles, and text line conditions $\mathbf{t}$ to improve alignment of characters. We further explain MCCBN in Section 3.2. Thanks to the convolutional properties of the generator, the receptive field of each character in the generated image overlaps with adjacent characters, which leads to natural transitions between characters and also enables generation of text in cursive.

The strategy explained above allows $\mathcal{G}$ to generate images of handwritten text with different sizes conditioned on character sequences $\mathbf{y}$ with arbitrary length, and with different styles that can be controlled by latent vector $\mathbf{z}$.

The loss function used to optimize $\mathcal{G}$ consists of two loss terms: adversarial loss, and text recognition loss. Adversarial loss is used to make $\mathcal{G}$ learn to generate images that $\mathcal{D}$ will consider as real images of handwriting. Text recognition loss is used to make $\mathcal{G}$ learn to generate real words that match the conditional input instead of gibberish.

Given character sequences $\mathbf{y}$ sampled from a lexicon of words, latent vectors $\mathbf{z}$ sampled from a normal distribution $\mathcal{N}(0, 1)$, and

text line conditions $\mathbf{t}$, the adversarial loss of $\mathcal{G}$ is

$$\mathcal{L}_{adv}^{\mathcal{G}} = -\mathbb{E}_{\mathbf{z}\sim p_z, \mathbf{y}\sim p_{data}} \mathcal{D}(\mathcal{G}(\mathbf{y}, \mathbf{z}, \mathbf{t})). \tag{3}$$

Text recognition loss of $\mathcal{G}$ is obtained by predicting text in generated images using $\mathcal{R}$:

$$\mathcal{L}_{CTC}^{\mathcal{G}} = \mathbb{E}_{\mathbf{z}\sim p_z, \mathbf{y}\sim p_{data}} [-\mathbf{y} \cdot \log \mathcal{R}(\mathcal{G}(\mathbf{y}, \mathbf{z}, \mathbf{t}))]. \tag{4}$$

Defining λ as a hyperparameter to balance the contribution of the two loss terms to the total loss, we optimize $\mathcal{G}$ by minimizing the joint loss term

$$\mathcal{L}_{\mathcal{G}} = \mathcal{L}_{adv}^{\mathcal{G}} + \lambda \cdot \mathcal{L}_{CTC}^{\mathcal{G}}. \tag{5}$$

All modules, i.e. $\mathcal{G}$, $\mathcal{D}$, and $\mathcal{R}$, are trained jointly. A simplified summary of the optimization process is presented in Algorithm 1, and an overview of our model is illustrated in 2.

Algorithm 1: Optimization process of the model

$\mathcal{G}, \mathcal{D}, \mathcal{R}$: Generator, Discriminator, Recognizer
I_R, I_G: real image, generated image
$\mathbf{y_R}, \mathbf{y_L}$: character sequences
$\mathcal{L}_{adv}^{\mathcal{G}}, \mathcal{L}_{adv}^{\mathcal{D}}$: adversarial loss functions
$\mathcal{L}_{CTC}$: CTC loss function
λ: weight of text recognition loss for $\mathcal{G}$ optimization
while *training* **do**
  sample I_R, $\mathbf{y_R}$ from Dataset
  sample $\mathbf{y_L}$ from Lexicon
  sample $\mathbf{z}$ from $\mathcal{N}(0, 1)$
  $\mathbf{t} \leftarrow$ get text line condition based on $\mathbf{y_L}$
  $I_G = \mathcal{G}(\mathbf{y_L}, \mathbf{z}, \mathbf{t})$
  update $\mathcal{D}$ to minimize $\mathcal{L}_{adv}^{D}(\mathcal{D}(I_R, I_G))$
  update $\mathcal{R}$ to minimize $\mathcal{L}_{CTC}(\mathcal{R}(I_R), \mathbf{y_R})$
  update $\mathcal{G}$ to minimize
   $\mathcal{L}_{adv}^{G}(\mathcal{D}(I_G)) + \lambda \cdot \mathcal{L}_{CTC}(\mathcal{R}(I_G), \mathbf{y_L})$
end

3.2 Multi-Class Conditional Batch Normalization

Class-conditional batch normalization [31] is a widely used method in GANs to generate images that belong to a specific class. In conventional class-conditional batch normalization, only one class is used for the entire image. However, in case of generation of character sequences, different regions in the feature maps require different class conditions that correspond to the characters in the sequence. We design a simple solution that allows us to use conditional batch normalization with multiple classes, which we call multi-class conditional batch normalization (MCCBN).

For conventional single-class conditional batch normalization

$$\hat{x} = \gamma^c \frac{x - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta^c, \tag{6}$$

the adaptive gain γ^c and bias β^c parameters are learned during network optimization and their values depend on the class condition c. Each class takes on a specific learned value for γ^c and β^c. In case of regular class conditional batch normalization, the conditional input is one class c and its corresponding γ^c and β^c are applied across the horizontal and vertical axes of the feature maps.

Figure 3: Illustration of the concept of baseline and mean line in the Latin alphabet.

In our case, we have a conditional input consisting of multiple classes $(y_1, ..., y_k)$ and their embeddings $(\mathbf{c}_1, ..., \mathbf{c}_k)$ which correspond to the characters in the target character sequence $\mathbf{y}$ of length k. Assuming that all characters have a similar width, we divide the feature maps into k regions of identical size and apply a different γ_i^c and β_i^c value to each feature map region. The γ_i^c and β_i^c values are dependent on class conditions $\mathbf{c}_i$, which are dependent on positions i in the target character sequence $\mathbf{y}$.

Let W be the horizontal size of the feature map. Multi-class conditional batch normalization can then be defined as:

$$\hat{x}_{mn} = \gamma_i^c \cdot \frac{x_{mn} - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta_i^c, \tag{7}$$

where m and n represent the horizontal and vertical positions in the feature map and

$$i = \left\lfloor m \cdot \frac{k}{W} \right\rfloor. \tag{8}$$

In other words, i is the position in a character sequence of size k.

In order to generate different handwriting styles, we also inject latent vectors $\mathbf{z}$ along with text line conditions $\mathbf{t}$ into MCCBN. We learn shared gain θ^z and bias δ^z which are dependent on a vector formed by concatenation of $\mathbf{z}$ and $\mathbf{t}$. We apply the same θ^z and δ^z across the whole feature map so that the style of handwriting is consistent. MCCBN with latent code injection, which we use in all our experiments, can then be defined as

$$\hat{x}_{mn} = \theta^z \cdot \gamma_i^c \cdot \frac{x_{mn} - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta_i^c + \delta^z, \tag{9}$$

3.3 Conditional Generation Based on Vertical Aspects of Text

Most scripts have a notion of baseline, which is a line on which the text rests. The Latin alphabet contains letters that reach below the baseline, such as 'g', 'p', or 'y'. Besides the baseline, there is also a notion of mean line in the Latin alphabet, which defines the height of most lowercase letters. Uppercase letters and some lowercase letters such as 'h' are taller and extend significantly above the mean line. Figure 3 shows an example describing the concept of baseline and mean line in text written in the Latin alphabet.

If a small case letter that extends only from the baseline to the mean line is generated across the entire height of the output image, any letter that extends below the baseline or above the mean line will not be able to naturally fit in the image and it will either cause misalignment of the baseline or the mean line, or it will be heavily distorted to compensate for the misalignment.

word examples	below baseline	above mean line
'common', 'woman'	x	x
'song', 'approve'	✓	x
'ball', 'another'	x	✓
'long', 'shortage'	✓	✓

Table 1: Examples of text line conditions (TLCs) for different words. There are four possible values of TLC based on the absence or presence of characters extending below the baseline or above the mean line.

To prevent misalignment and distortion of the generated word images caused by the lack of awareness of the baseline and mean line in the target word, we introduce an additional condition in the input of the generator. The condition has four possible values depending on the presence or absence of letters that extend below the baseline or above the mean line in the target character sequence $\mathbf{y}$ as shown in Table 1. We embed this condition into a vector $\mathbf{t}$ which we concatenate with the latent vector $\mathbf{z}$. For simplicity, we further refer to this conditioning as text line condition (TLC).

We only use TLC for experiments on the Latin alphabet data because Japanese text does not have a similar notion of multiple alignment lines in the text line and all characters rest on the baseline.

4 EXPERIMENTS

4.1 Implementation Details

The architecture of $\mathcal{G}$ and $\mathcal{D}$ is based on BigGAN [7], with CBN layers replaced by our proposed MCCBN. The architecture of $\mathcal{R}$ is based on [35] without recurrent layers. The model is optimized using the Adam optimizer [22] with a learning rate of 0.0002 and $(\beta 1, \beta 2) = (0.0, 0.999)$. We set λ in the loss function of $\mathcal{G}$ to 1. During training, we group image samples into batches by the word length so that all images in each batch have the same size. For experiments on the Latin alphabet datasets, the size of latent vector $\mathbf{z}$ is 160, and the vector is split into 5 chunks of size of 32, which are passed to each level of $\mathcal{G}$ and the base filter f. The size of character embeddings is set to 32 to match the size of $\mathbf{z}$. Unless specified otherwise, the size of text line condition embedding $\mathbf{t}$ is 4. We conduct the experiments on a single NVIDIA V100 GPU and we set the batch size to 8.

4.2 Datasets

We use the following datasets for our experiments:

- **IAM**. The IAM dataset [29] consists of about 80k images of handwritten English words. It is divided into training, test, and two validation sets. The training set contains about 40k images. The words are written by 657 different authors, and all words written by each author only appear in one of the four sets to provide sets with mutually exclusive authors.
- **CVL**. The CVL dataset [24] contains handwritten manuscripts written by 311 different writers, each of whom wrote 6 different documents in English. The total number of cropped words in the dataset adds up to 83k.
- **Simulated Japanese handwriting dataset**. For Japanese, we use 61 free handwriting fonts [1] to create a dataset of 1M images that simulates handwritten text written by 61 different writers. The dataset contains 2136 kanji (Chinese characters) and 2 Japanese syllabaries, adding up to 2302 character classes in total.

We resize all images to a fixed height of 32 pixels while keeping their original aspect ratio.

The lexicons employed in our experiments are extracted from the used datasets and we do not use additional text corpora.

4.3 HTR Evaluation Metrics

Following [12], we use a state-of-the-art text recognition network from [5] to train HTR models. Our HTR models are trained from scratch using labeled data from the IAM dataset, and also data produced by trained generative models. We employ two standard metrics to evaluate the performance. Word error rate (WER) indicates the percentage of misread words in the test set. Normalized edit distance (NED) is the edit distance between the predicted word and the target word normalized by the length of the target word.

4.4 Data Augmentation for HTR

Collecting labeled data is much more difficult than collecting data without annotation. As a result, in real life situations, we may have access to data which is only partially annotated or we may be able to collect annotations only for a subset of our unlabeled data due to the labor cost required to make annotations. In our model for handwritten text generation, only the text recognition module requires labeled data for its optimization. The discriminator only predicts whether an image of handwritten word is realistic or not, so we can use unlabeled data to optimize it. This allows us to train our model with only partially labeled data. To simulate the situation where we have only partially labeled data, we create IAM-5k dataset consisting of $\frac{1}{8}$ of the data from IAM, which is about 5k labeled images. The rest of the images from IAM is used as unlabeled data. Then, we train our model in a semi-supervised fashion using both the labeled data from IAM-5k and the unlabeled data.

As a baseline for evaluation, we employ an HTR model trained only on IAM-5k. Then, we augment the IAM-5k dataset with 100k images produced by our conditional generative models trained in a semi-supervised fashion, and train HTR models on these augmented datasets. The results presented in Table 2 demonstrate that augmenting the training dataset for HTR with images generated by our model trained using both labeled and unlabeled data improves the HTR performance. In addition, HTR models trained on datasets augmented with images generated by our proposed model achieve better results than HTR models trained on a dataset augmented with images generated by existing state-of-the-art model for handwritten text generation.

Table 3 shows that we can achieve boost in HTR performance even when we employ the models trained in the way explained above on a different dataset, which may be helpful when we want to perform HTR on data that we have no annotation for. We use the test set of the CVL dataset to demonstrate this.

[1] https://www.freejapanesefont.com/category/handwriting/

Training Data	WER ↓	NED ↓
IAM-5k (baseline)	55.56	27.07
IAM-5k + ScrabbleGAN [12] 100k	38.97	14.35
IAM-5k + JokerGAN w/o TLC 100k	38.43	14.18
IAM-5k + JokerGAN 100k	**36.30**	**13.04**
IAM (oracle)	16.52	4.94

Table 2: Text recognition evaluation on the test set of IAM dataset using HTR models trained on data specified in the table. We use an HTR model trained on 5k labeled images as the baseline. Models that are trained on datasets augmented by 100k images generated by JokerGAN trained in semi-supervised manner on labeled and unlabeled data yield better performance than the baseline. Models trained on datasets augmented by JokerGAN also outperform those trained on data augmented by existing state-of-the-art. The last row shows performance of an HTR model trained on the full IAM dataset of approximately 40k labeled images.

Training Data	WER ↓	NED ↓
IAM-5k (baseline)	77.39	47.59
IAM-5k + ScrabbleGAN [12] 100k	67.01	32.36
IAM-5k + JokerGAN w/o TLC 100k	65.47	31.53
IAM-5k + JokerGAN 100k	**62.33**	**28.42**

Table 3: Text recognition evaluation on the test set of CVL dataset using HTR models trained on data specified in the table. The results show that data augmentation is effective even if we apply the trained HTR model on a different dataset.

Method	FID ↓	GAN-train ↓	GAN-test ↓
ScrabbleGAN [12]	14.31	51.84	28.41
JokerGAN w/o TLC	10.37	49.88	14.78
JokerGAN (TLC, size 1)	11.12	55.09	11.81
JokerGAN (TLC, size 4)	**9.18**	**49.14**	**10.90**
JokerGAN (TLC, size 8)	10.87	55.47	15.60

Table 4: Evaluation of our proposed model against the state-of-the-art using GAN evaluation methods. For GAN-train and GAN-test tests, we use WER as the measurement of performance. We also train our model without text line conditions (TLC) and with different TLC embedding sizes to evaluate the performance of TLC.

4.5 GAN Metric Evaluation

Following existing work, we use Frechet inception distance (FID) [17] to evaluate performance of the generative model. FID can measure visual quality and sample diversity, but it was introduced for unconditional image generation and it cannot tell us how well the results match the conditions. To address that, we employ GAN-train and GAN-test metrics [36], which evaluate conditional image generation on a downstream image recognition task. For GAN-train, a recognition model is trained on generated images and tested on a test set of real images. For GAN-test, real images are used to train a model which is then tested on generated data. GAN-train is an indicator of diversity of generated images, and GAN-test is a measure of fidelity of generated images with respect to the original data. We use HTR as the downstream task and WER to measure recognition performance. As can be seen in Table 4, our proposed model achieves superior results in all metrics. The results indicate that images generated using our proposed model have significantly higher fidelity, in particular.

Method	User preference
ScrabbleGAN [12]	26.5
JokerGAN (w/o TLC)	29.6
JokerGAN	**43.9**

Table 5: User preference study. The numbers show the percentage of people who find the images generated by the respective methods more realistic and natural. We used the IAM dataset to train the models.

4.6 Text Line Conditioning

Our GAN metric evaluation, presented in Table 4, indicates that using text line conditions (TLC) improves the quality of generation. The size of TLC embeddings, however, affects the performance. We conduct experiments with different embedding sizes and we empirically find that the optimal size in our settings is 4, as can be seen in Table 4.

Samples of generated images in Figure 4 indicate that using TLC leads to better alignment of characters. Notice particularly that characters extending below the baseline or above the mean line in images generated by JokerGAN with TLC, shown in the leftmost column, look more realistic and do not suffer from misalignment or distortion.

4.7 Human Evaluation

We conduct a study on Amazon Mechanical Turk (AMT) to compare the visual fidelity of images generated by our proposed model against the state-of-the-art method. The AMT workers are given a task to select the most realistic looking image out of three images of the same word generated by three different methods. We randomly generate 100 questions and each question is answered by 20 different workers.

Table 5 shows the evaluation results. We find that users perceive images generated by our proposed model more realistic than images produced by the existing state-of-the-art method. Furthermore, the results also show that users strongly favor the images generated by the model trained with text line conditioning.

4.8 Qualitative Evaluation

Figure 4 shows a comparison of images generated by JokerGAN and images generated by the existing state-of-the-art method. Images generated by JokerGAN look more refined and have more natural shapes, particularly when using text line conditioning.

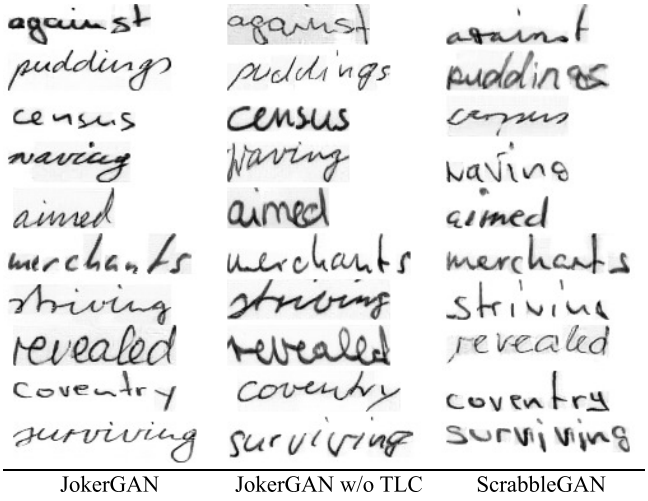

Figure 4: The first column shows images generated by JokerGAN. The second column shows images generated by JokerGAN without text line conditioning. The third column shows images generated by ScrabbleGAN [12].

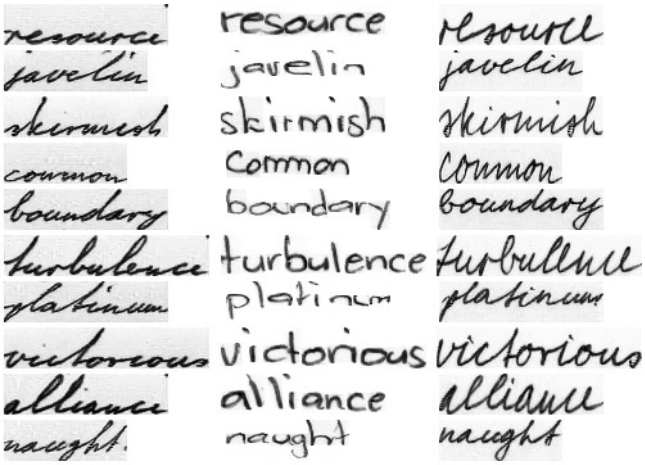

Figure 5: Generating different styles with JokerGAN. Images in each column are generated using the same latent vector with different word conditions.

In Figure 5, you can see images generated by our proposed method with fixed latent vectors. Each column shows images generated with the same latent vectors and different word conditions. When using the same latent vectors, the style of generated images remains very similar even when we change the word condition.

Figure 6 shows examples of images generated by JokerGAN trained on our Japanese dataset, and their corresponding target labels. Despite the large number of classes in the dataset, our model is able to generate images of text that match the input conditions and look highly plausible in most cases.

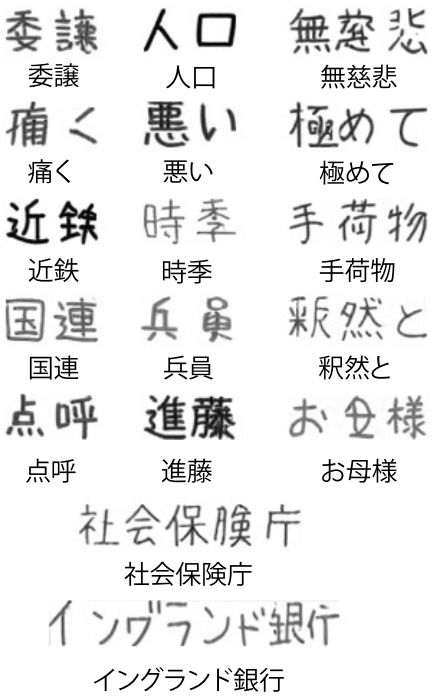

Figure 6: Examples of generated images of Japanese handwritten words conditioned by character sequences shown below each image.

Method	Latin alphabet	2302 Japanese chars
ScrabbleGAN [12]	23.1M	606.2M
JokerGAN	2.8M	3.6M

Table 6: Comparison of the number of parameters in the generator of JokerGAN and ScrabbleGAN depending on the size of the character set in the used dataset.

4.9 Model Size

Table 6 shows a comparison of the number of parameters in the generator of JokerGAN and ScrabbleGAN. As you can see, despite the large difference in the number of characters when training either on the Latin alphabet or Japanese characters, the size of our model does not significantly change. On the other hand, the size of the ScrabbleGAN generator grows nearly linearly with respect to the number of characters.

5 CONCLUSION

We have proposed a novel architecture for generation of handwritten text images. In contrast to the existing state-of-the-art method for generation of handwritten text images, whose memory requirements grow significantly as the number of characters in the character set increases, the memory requirements of our model remain virtually the same regardless of the size of the character set in the training dataset. This allows us to train our model not only on

the Latin alphabet, but also on datasets with large character sets such as Japanese. Furthermore, while boasting lower memory requirements, our proposed method also achieves better performance when applied as a data augmentation method for HTR systems. We have also introduced a new type of conditioning which makes the generator aware of vertical properties of characters in the target word and improves the alignment of characters in the generated image. A human evaluation study further confirms that our method can generate more realistic images of handwritten text.

REFERENCES

[1] Francesco Adamo, Filippo Attivissimo, Attilio Di Nisio, and Maurizio Spadavecchia. 2015. An Automatic Document Processing System for Medical Data Extraction. *Measurement* 61 (2015), 88–99.

[2] Emre Aksan, Fabrizio Pece, and Otmar Hilliges. 2018. DeepWriting: Making Digital Ink Editable via Deep Generative Modeling. In *CHI*.

[3] Eloi Alonso, Bastien Moysset, and Ronaldo Messina. 2019. Adversarial Generation of Handwritten Text Images Conditioned on Sequences. In *ICDAR*.

[4] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. 2016. MedRec: Using Blockchain for Medical Data Access and Permission Management. In *2016 2nd International Conference on Open and Big Data (OBD)*.

[5] Jeonghun Baek, Geewook Kim, Junyeop Lee, Sungrae Park, Dongyoon Han, Sangdoo Yun, Seong Joon Oh, and Hwalsuk Lee. 2019. What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis. In *ICCV*.

[6] Claus Bahlmann, Bernard Haasdonk, and Hans Burkhardt. 2002. Online handwriting recognition with support vector machines - a kernel approach. In *Proceedings Eighth International Workshop on Frontiers in Handwriting Recognition*. 49–54.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. 2018. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *ICLR*.

[8] Bo Chang, Qiong Zhang, Shenyi Pan, and Lili Meng. 2018. Generating Handwritten Chinese Characters using CycleGAN. In *WACV*.

[9] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. 2017. Disease prediction by machine learning over big data from healthcare communities. *IEEE Access* 5 (2017), 8869–8879.

[10] Brian Davis, Chris Tensmeyer, Brian Price, Curtis Wigington, Bryan Morse, and Rajiv Jain. 2020. Text and Style Conditioned GAN for Generation of Offline Handwriting Lines. In *BMVC*.

[11] Kartik Dutta, Praveen Krishnan, Minesh Mathew, and CV Jawahar. 2018. Improving CNN-RNN Hybrid Networks for Handwriting Recognition. In *ICFHR*.

[12] Sharon Fogel, Hadar Averbuch-Elor, Sarel Cohen, Shai Mazor, and Roee Litman. 2020. ScrabbleGAN: Semi-Supervised Varying Length Handwritten Text Generation. In *CVPR*.

[13] Ji Gan and Weiqiang Wang. 2021. HiGAN: Handwriting imitation conditioned on arbitrary-length texts and disentangled styles. In *AAAI*.

[14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. In *NIPS*.

[15] Alex Graves. 2013. Generating Sequences With Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850* (2013).

[16] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML*.

[17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *NIPS*.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[19] Jianying Hu, Michael K Brown, and William Turin. 1996. HMM based online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18, 10 (1996), 1039–1045.

[20] Bo Ji and Tianyi Chen. 2019. Generative Adversarial Network for Handwritten Text. *arXiv preprint arXiv:1907.11845* (2019).

[21] Lei Kang, Pau Riba, Yaxing Wang, Marçal Rusiñol, Alicia Fornés, and Mauricio Villegas. 2020. GANwriting: Content-Conditioned Generation of Styled Handwritten Word Images. In *ECCV*.

[22] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[23] Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

[24] Florian Kleber, Stefan Fiel, Markus Diem, and Robert Sablatnig. 2013. CVL-DataBase: An Off-Line Database for Writer Retrieval, Writer Identification and Word Spotting. In *ICDAR*.

[25] Jae Hyun Lim and Jong Chul Ye. 2017. Geometric GAN. *arXiv preprint arXiv:1705.02894* (2017).

[26] Xiyan Liu, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2019. FontGAN: A Unified Generative Framework for Chinese Character Stylization and Destylization. *arXiv preprint arXiv:1910.12604* (2019).

[27] Zichuan Liu, Yixing Li, Fengbo Ren, Wang Ling Goh, and Hao Yu. 2018. SqueezedText: A Real-Time Scene Text Recognition by Binary Convolutional Encoder-Decoder Network. In *AAAI*.

[28] Nam-Tuan Ly, Cuong-Tuan Nguyen, Kha-Cong Nguyen, and Masaki Nakagawa. 2017. Deep Convolutional Recurrent Network for Segmentation-Free Offline Handwritten Japanese Text Recognition. In *ICDAR*.

[29] U-V Marti and Horst Bunke. 2002. The IAM-database: an English sentence database for offline handwriting recognition. *IJDAR* 5, 1 (2002).

[30] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. *arXiv preprint arXiv:1411.1784* (2014).

[31] Takeru Miyato and Masanori Koyama. 2018. cGANs with Projection Discriminator. In *ICLR*.

[32] Rejean Plamondon and Sargur N Srihari. 2000. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 1 (2000), 63–84.

[33] Thomas Plötz and Gernot A Fink. 2009. Markov models for offline handwriting recognition: a survey. *IJDAR* 12, 4 (2009), 269.

[34] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[35] Baoguang Shi, Xiang Bai, and Cong Yao. 2016. 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 (2016), 2298–2304.

[36] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. 2018. How good is my GAN?. In *ECCV*.

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

[38] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. *arXiv preprint arXiv:1409.3215* (2014).

[39] Shan-Jean Wu, Chih-Yuan Yang, and Jane Yung jen Hsu. 2020. CalliGAN: Style and Structure-aware Chinese Calligraphy Character Generator. *arXiv preprint arXiv:2005.12500* (2020).

[40] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *ICCV*.

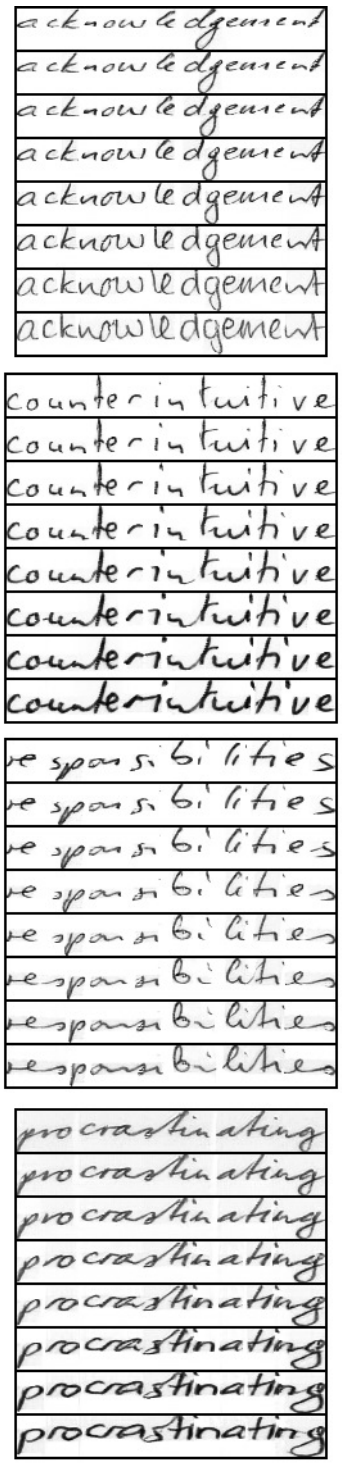

Figure 7: Examples of style interpolation between two different styles defined by two different latent vectors.

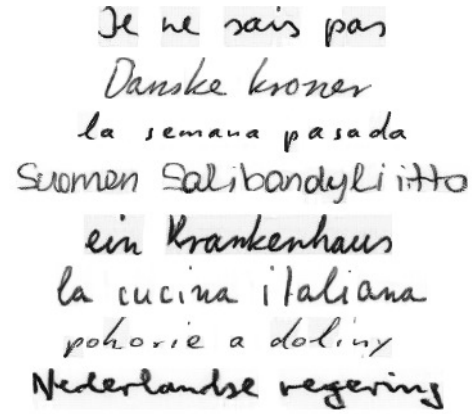

Figure 8: Examples of generated images of out-of-vocabulary words and phrases. We use a generative model trained on the IAM dataset of English words to generate handwritten text in different languages that also use the Latin alphabet. Starting from the top, the languages are French, Danish, Spanish, Finnish, German, Italian, Slovak, and Dutch.

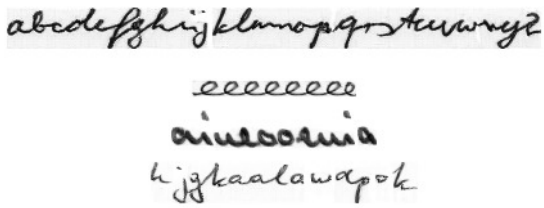

Figure 9: Examples of out-of-vocabulary generation. Since our model uses character-level conditioning, we can generate images of arbitrary character sequences, including non-existent words and random character sequences.

A STYLE INTERPOLATION

In Figure 7, we show examples of interpolation between two different styles defined by two different latent vectors. As can be seen, the style gradually changes from one style to another when generating images using interpolated latent values. This indicates that our model can generalize on the handwriting styles and continuous changes in the latent vector results in images whose appearance changes continuously.

B GENERATION OF OUT-OF-VOCABULARY PHRASES

Since our model uses character-level conditioning, we can feed arbitrary character sequences as conditions into our generator. This allows us to generate words that are not in the dataset used for training. Despite our model being trained only on a dataset of English words, we can generate images of handwritten words in other languages that also use the Latin alphabet, as demonstrated in Figure 8. However, since our model is trained on the character set used in English, we cannot generate words with special characters such as letters with diacritics, e.g. words like 'école', 'voilà', and 'tête'

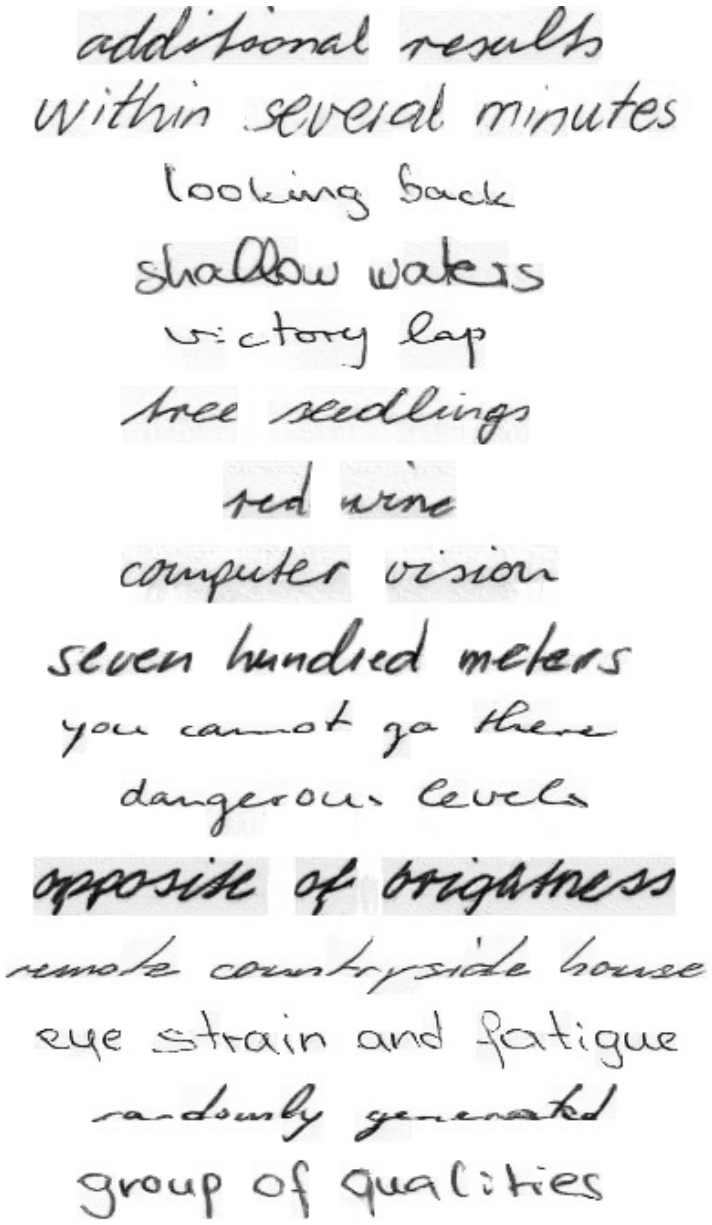

Figure 10: Examples of generated images of phrases comprising of multiple words. The model that generated these images was trained only on data consisting of single handwritten words. The presented examples are chosen randomly without cherry-picking.

in French. We can also generate completely non-existent words and random character sequences, as illustrated in Figure 9.

C GENERATION OF MULTIPLE WORD PHRASES

We train JokerGAN on datasets of handwritten words; therefore, there are no whitespace characters in our training images. We demonstrate that we can train our model to generate whitespace even when training on datasets of single handwritten words. To achieve this, we randomly add whitespace of the size of one character either on the left side or right side of the image, and add the whitespace character label at the beginning or the end of the sequence of labels corresponding to the word in the image. A model trained using this approach can generate not only single words but also long phrases comprising of multiple words as illustrated in Figure 10.

Figure 11: Examples of visual questions given to workers on Amazon Mechanical Turk. In each question, there is an image generated by JokerGAN, an image generated by JokerGAN without text line conditioning, and an image generated by ScrabbleGAN. Workers have to select the most natural looking image.

D EXAMPLES FROM HUMAN STUDY EVALUATION

In Figure 11, you can see examples of visual questions given to workers on Amazon Mechanical Turk. The workers have to select the image of a handwritten word that looks the most realistic to them. There is a caption provided for each question stating what word is supposed to be written in the images. For each question, there are three images of a handwritten word. One of them is generated by JokerGAN, one by JokerGAN trained without text line conditioning, and one is generated by ScrabbleGAN. The order of the three images is randomized.

E ADDITIONAL EXAMPLES

Figure 12 shows additional, randomly selected examples of handwritten word images generated by JokerGAN.

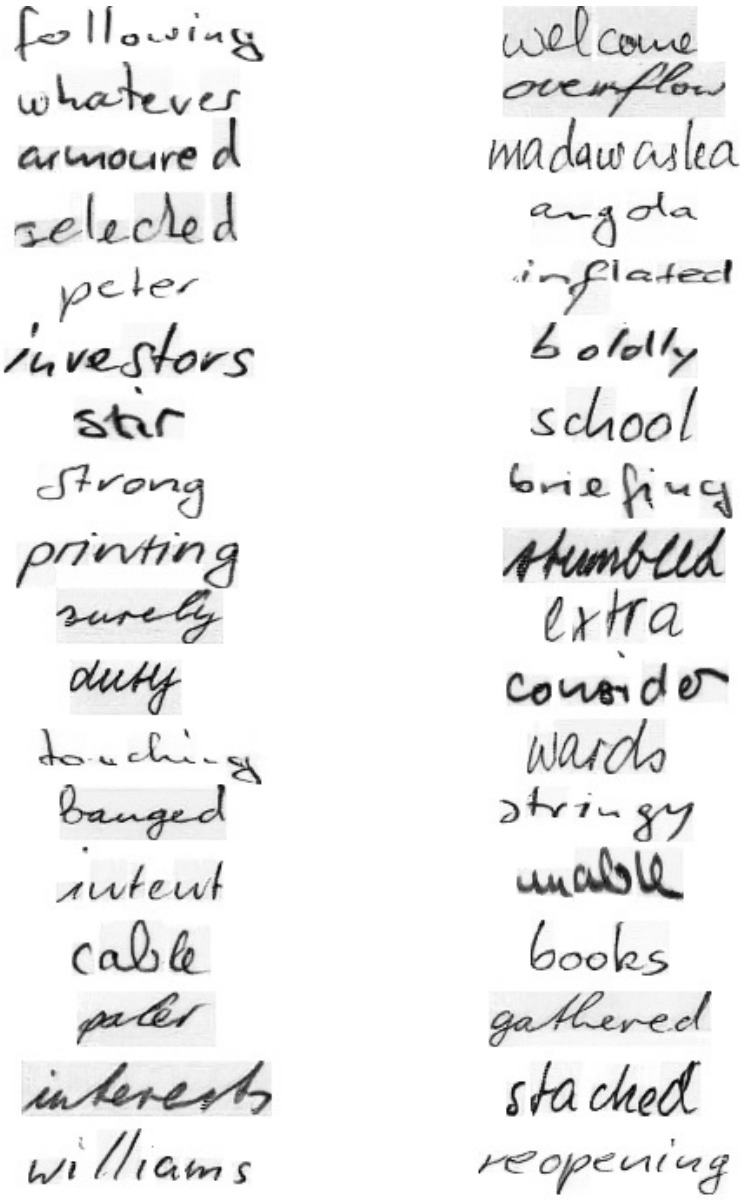

Figure 12: Additional examples of handwritten word images generated by JokerGAN. The examples are chosen randomly without cherry-picking.