university of groningen / faculty of science and engineering

Image-to-Image Translation for Handwriting Generation using GANs

Bachelor's Project Thesis

T.P.J. Bakker, t.p.j.bakker@student.rug.nl,
Supervisors: Prof. Dr. L.R.B Schomaker & S. Rezaee

Abstract: In this paper it is investigated whether machines are able to synthesize human-like handwriting through deep learning that can be used to augment datasets. For this purpose the relatively novel generative adversarial networks (GANs) are deployed and experimented with. GANs are generative neural networks, consisting of two competing networks, that adversarially learn through a loss function to generate new, non-existing samples. There exist several potentially useful loss functions such as L1 and L2. By feeding the network images with machine-print words and simultaneously feeding it images with handwritten words the system learns to mimic the handwriting. These synthesized handwritten words can be used to supplement datasets used to train Handwritten Text Recognition (HTR) systems. Several experiments demonstrate what the optimal settings are in terms of network architecture to achieve machine-print handwriting resembling human writing.

1 Introduction

Handwritten Text Recognition (HTR) is a domain in pattern recognition that has been extensively researched over the past decades. The ability to translate physically written text to a digital representation opened up possibilities in different areas of expertise. For example, in forensics HTR could serve as a means of matching evidence to a suspect (Bulacu, 2007). Convolutional neural networks (CNNs) in combination with Long Short-Term Memory recurrent neural networks (LSTMs) (Hochreiter and Schmidhuber, 1997) are frequently implemented in these offline systems that serve the purpose of recognizing text drawn by pen or pencil.

Correspondence, however, is nowadays shifting towards being primarily digital while classical textual communication is in decline. People are more inclined to convey a message by e-mail or by one of the several popular cross-messaging platforms. Although these message services provide an increase in convenience in terms of speed, with respect to more traditional ways of communication, the personal aspect of handwritten text is lost.

One way to not entirely shut down the door on handwritten script is to implement a neural network that is able to generate text conform to personal handwriting. For such a system to work one would need to represent text in a manner that can be processed by the network. Various research has resulted in models suitable for this purpose, such as the Skip-gram model (Mikolov, Sutskever, Chen, Corrado, and Dean, 2013). This model creates vector representations of words and captures syntactic and semantic relationships between words in a text. Another frequently used method is the n-grams model (Brown, Desouza, Mercer, Pietra, and Lai, 1992) by which words are statistically predicted based on previous words. Handwriting generation

Figure 1.1: Variability in how the character 'p' is written in the words 'sleep', 'pull' and 'spare'. Meticulous details depend on the neighbours of characters. A system that mimics these words should be able to distinguish between such varieties.

rather focuses on the relationship between individual characters than on the relation between multiple words. The way characters are written in a word is often dependent on the relationship with their neighbouring characters, as shown in Figure 1.1. A possible solution to this problem is to represent a word as character bigrams: all two-character combinations in a single word. Many words, however, have predominantly equal bi-gram representations leading to an impoverished data distribution.

To overcome the restrictions of having a sparse data distribution in handwriting synthesis *image-to-image translation* is used in this study. Image-to-image translation is referred to as the mapping of a set of (input) pixels to a set of (output) pixels (Isola, Zhu, Zhou, and Efros, 2017). If the set of input pixels is an image containing a word written as machine print and the set of output pixels is the same word but in handwritten fashion then the data distribution is richer than the bigram distribution.

The missing piece of the puzzle is the generative system itself. Generative Adversarial Networks (GANs), the recently proposed neural networks (Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, and Bengio, 2014), provide this remaining piece. GANs consist of two separate neural networks, a discriminator and a generator. The generator aims to fool the discriminator by synthesizing data which the discriminator classifies as real (i.e. coming from the training data) or fake (i.e. produced by the generator). Alongside GANs there is another theoretical option for implementing the generative system: Variational Autoencoders (VAEs) (Kingma and Welling, 2013). VAEs, however, produce blurry images in contrast to GANs (Radford, Metz, and Chintala, 2015). Characters in words are subject to detail which blurring likely fails to capture, therefore GANs have the upper hand.

In this study, GANs are researched with the purpose of generating handwritten words through image-to-image translation. This specialised type of GAN will be termed *GANdwriter*. A succesful implementation of the GANdwriter has the ability to generate data that can be used to augment training data sets. Given the fact that such a system is conditioned to synthesize handwriting corresponding to an input image, the correct labels are automatically included.

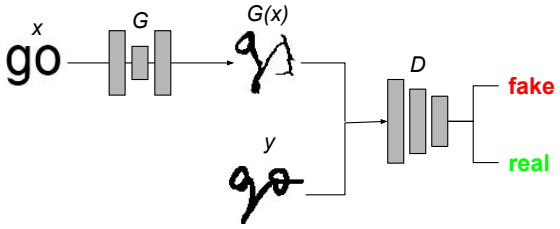

Figure 2.1: Procedure of training the GAN: G translates the input image x to a fake output image through $G(x)$. D learns to discriminate between the fake and real images.

2 Methods

The goal of traditional GANs is to learn how to map a random noise vector, z, to an output image, y, using a generator, $G : z \rightarrow y$ (Goodfellow et al., 2014). In these networks the generator G is adversarially trained with a discriminator D. The procedure of training GANs is characterized by G and D playing a minimax two-player game: G is generating images with the purpose of 'fooling' D as D is classifying images as 'fake' (i.e. produced by the generator) or 'real' (i.e. being a sample from the training data). Classification between fake and real images is expressed as the probability of an image being fake or real. Successful training implies that G is able to mimic the images from the training data distribution while D converges to a probabilistic equilibrium: $p_{generated} = p_{data}$ (Goodfellow et al., 2014).

In this study, the input random noise vector z is replaced by an input image x so that $G : x \rightarrow y$. On each image x a label is placed in machine-print space, corresponding to the word in sample y of the training distribution, in handwritten space. This way G is effectively conditioned to translate the input word to the same word but in handwritten manner. Figure 2.1 is the graphical representation of the training process.

2.1 GANs objective

The objective of GANs, where G aims to minimize the loss function while D aims to maximize it, can

be expressed as

$$\min_G \max_D L_{GAN}(G, D) = \mathbb{E}_y[\log D(y)] \; + \; \mathbb{E}_x[\log(1 - D(G(x)))]. \quad (2.1)$$

The task of G is not only to fool D but also to synthesize images that closely resemble the ground truth of images in the training distribution. One way to decrease blurring and thereby facilitate an increase in precise resemblance is to introduce the mean absolute error (i.e. *L1*) (Isola et al., 2017):

$$L_{L1}(G) = \mathbb{E}_{x,y}[|y - G(x)|], \quad (2.2)$$

leading to the final objective for G:

$$G(\theta_G) = L_{GAN}(G, D) + \lambda L_{L1}(G) \quad (2.3)$$

where θ_G are the parameters of G; λ is a constant for controlling the amount of contribution of L1. In research of (Isola et al., 2017) $\lambda = 100$ has been found to be most effective and is therefore adopted. An alternative to L1 would be the mean squared error (i.e. *L2*). According to past research L2 suffers from more blurring with respect to L1 (Zhao, Gallo, Frosio, and Kautz, 2016). Because the results preferably have the least amount of blurring as possible, the focus lies on the L1 variant.

2.2 Network architecture

The settings of the hyperparameters of GANs are sensitive in a sense that small adjustments to the architecture of a network may result in drastic differences in its final output. (Salimans, Goodfellow, Zaremba, Cheung, Radford, and Chen, 2016) researched several improvements to (partially) overcome this sensitivity. One of the proposed solutions is to introduce *minibatch discrimination*: multiple images are presented to the discriminator in a single training step. Minibatch discrimination removes the phenomenon of a GAN collapsing into a mode where all generated outputs come from a single point. (Masters and Luschi, 2018) investigated the sweet spot for the size of such a minibatch. They have found that a minibatch size m, where $2 \geq m \leq 32$, achieves the best results in a consistent way. In the architecture of the GANdwriter $m = 32$ is chosen because the samples in the dataset show high variability at the character level. By presenting more samples in a training step the network better adapts to this variability. Moreover, the distribution of the number of samples per word is not normal: some words have significantly more samples than other words. Therefore, a higher minibatch size decreases the chance that a batch contains only samples of a single word, which would disfavour the training process.

To overcome the well-known problem of overfitting, *dropout* layers are added to the architecture of both networks, G and D (Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov, 2014; Wager, Wang, and Liang, 2013). Dropout layers shut down a random portion of its input units thereby diminishing the effect of several units becoming overly influential in the output of both networks.

For optimization purposes, *batch normalization* layers are added to each block in both networks, except for the first block (Ioffe and Szegedy, 2015). Batch normalization is used to reduce time needed for a network to converge by normalizing the input distribution of a layer, which diminishes the internal covariate shift.

Minibatch stochastic gradient descent is used in the form of the Adam optimizer (Kingma and Ba, 2014). Both the generator and the discriminator have an individual optimizer with equivalent parameters: a learning rate of 2 x 10^{-4}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The gradients are applied alternately: first a step on the discriminator, then a step on the generator (Goodfellow et al., 2014). Furthermore, as suggested by (Goodfellow et al., 2014; Isola et al., 2017) the generator is trained to maximize $\log D(G(x))$ rather than to minimize $1 - \log D(G(x))$. Maximizing the former loss function provides stronger gradients in the early stages of learning where the discriminator can quickly become too powerful. In this early stage the generated images can be confidently classified as false by the discriminator because they do not resemble the real images yet.

Architecture details are included in Appendix A.

2.2.1 PatchGAN

An important feature of the traditional GAN is that the traditional discriminator outputs a single scalar p, where $-1 \geq p \leq 1$. This represents the probability of the output being real (i.e. $p = 1$)

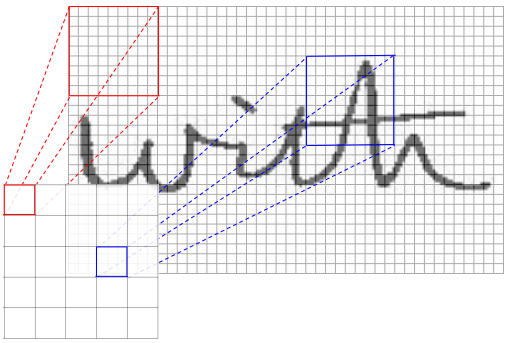

Figure 2.2: PatchGAN discriminator. Each cell in the 5x5 grid (output of the discriminator) classifies whether a patch of 9x9 on the original image is real or fake. This is merely an illustrative example, as patch size and output grid depend on the details of the convolutional layers used in the discriminator.

or fake (i.e. $p = -1$). Such a classification fails to penalize local features in the output of the generator, because the generator only receives feedback from the discriminator on the difference between the entire images. An improvement to this problem is the *PatchGAN* (Isola et al., 2017). This type of discriminator outputs an NxN grid of which each cell classifies an MxM patch of its input images. A cell in this NxN grid has a receptive field (i.e. the MxM patch) (Dumoulin and Visin, 2016), which is averaged through convolution. In this way the attention is focused on local features rather than on global features. Figure 2.2 depicts this patch classification.

The patch-size is the result of the recursive application of the formula that calculates a network's layer local receptive field, which finally yields the global receptive field. To calculate the global receptive field (i.e. the patch size)

$$RF_{out} = RF_{in} + (k-1) * s \tag{2.4}$$

is recursively used, where RF is the receptive field, k is the kernel size and s is the cumulative stride. At each recursive step s is multiplied by the current layer's stride. Initially, the values of RF_{in} and s are equal to 1.

3 Experiments

To test what the optimal network-settings of the GANdwriter are, several experiments have been performed. All of the following results are generated by networks trained on the same dataset. This dataset contains images of varying dimensions with single words in English, labelled, in handwriting space. The images were all resized to a uniform size (256 x 64) respecting original character shapes and character dimensions as well as possible. Originally, the dataset contained 60,719 images with a great variability in word length which leads to great variability in the dimensions of the images. To speed up the training process by restricting images to have maximum dimensions, the dataset was reduced to words of word length w to $1 \geq w \leq 5$, resulting in a dataset size of 25,110 images. Furthermore, all images were normalized to $[-1, 1]$ to optimize processing: the goniometric formulae used benefit from this interval of values. Results were gathered after training the models for 20 epochs on a Tesla K80 GPU (Google Colab), which took approximately 2 hours. The models were implemented using the TensorFlow framework.

Because the input images in this study are not square but rather rectangular, the actual discriminator output grid is also rectangular. The receptive field, however, is square as in figure 2.2 due to the properties of convolution.

3.1 Evaluation

It is well known that evaluating GANs is a difficult problem for which no consensual solution has been constituted (Borji, 2019; Salimans et al., 2016). In HTR, for example, a system is scored on accuracy in terms of how well words or characters are classified. Accuracy is calculated with respect to the training dataset: it is known when a word is classified correctly or incorrectly. In handwriting generation, however, novel samples are synthesized. These new samples are different from samples in the training data distribution and can therefore not easily be compared to the training samples as is done in HTR.

There do exist evaluation metrics for handwriting generation systems, both qualitative as well as quantitative. A qualitative analysis can be performed through human classification: persons are

Figure 3.1: Performance comparison between different loss functions. Each column shows the results using L1, L2 or traditional loss compared to the ground truth.

presented words which they have to classify as real or fake. The percentage of people classifying the generated words as real then serves as an indication of how well the system is able to mimic handwriting. There is a drawback to this method of evaluation, however. The problem is that the used loss functions in a system do not necessarily predict human classification. L1 might converge to some global optimum that is an average of the variability in a dataset. This does not mean that this global optimum corresponds with what we humans would perceive as believable handwriting. In contrast, an HTR system is a direct solution for this problem. Generated words can be presented to an existing, pretrained model that recognizes and classifies handwritten text. If this model is able to recognize the synthesized words then the generative model can be regarded as successful. This pseudo-metric has been proposed in earlier work (Isola et al., 2017; Salimans et al., 2016) and is a way of quantitative evaluation. Random examples generated by the GANdwriter are shown in figure 4.1.

3.2 Analysis of the objective

Figure 3.1 shows the qualitative results that are obtained using the GANdwriter with different loss functions. The resulting words synthesized by using only the traditional GAN loss (Equation 2.1) are all more or less equivalent. This is an example of a GAN collapsing into a single mode where all outputs are similar. L2 in combination with the GAN loss produces more believable results with respect to the traditional objective but with the predicted blurring. Moreover, the background contains a slight amount of noise. The best results, in terms of crispness, are achieved by L1 in combination with the traditional loss function.

3.3 Analysis of the PatchGAN

Some tests were performed to investigate the influence of the receptive field size in the discriminator output. The results are shown in figure 3.2. Using an 18x18 patch achieves the most structured output: most characters resemble the characters in the ground truth. Furthermore, relative to the 38x38 patch, characters are more fluently connected with a fewer amount of seemingly random lines. The 78x78 patch example shows that using a greater

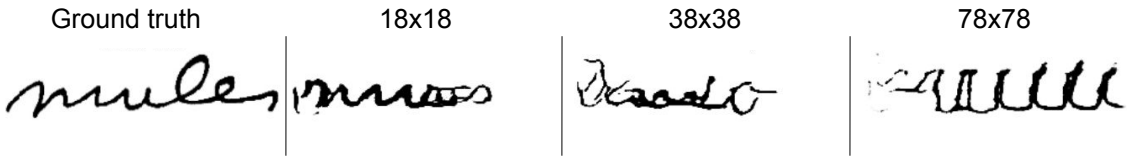

Figure 3.2: Effect of the patch size on output quality. The 18x18 **patch achieves the most similar results. A patch size greater than the input image height causes the GAN to collapse into a single mode.**

height in the patch than the input image height results in a single mode collapse. A reason for this is that the words on the images are placed on the left center side and are padded with white space on the right and equally on top and bottom. Therefore, most of the images contain significantly more white space than handwriting pixels. Larger patch sizes capture less features in this case as larger receptive fields correspond with a smaller discriminator output grid.

3.4 Future work

The current GANdwriter focuses on the general structure of words as a whole because most individual characters are conjoined cursively and therefore there is no clear separation between them. A drawback of this method is that there is no explicit structural attention to individual characters while language and handwriting is per definition structured. Most characters in a certain position in a word are often written identically: the 'n' in 'and', for example, is written similarly every time it occurs in that configuration of neighbouring characters. Therefore, it would be interesting to investigate the possibility of focusing on individual characters in cursive handwriting. Block letter handwriting is less intricate with respect to cursive writing and could also serve as a starting point for future research.

Conditional GANs (cGANs) are a subclass of traditional GANs (Isola et al., 2017). cGANs extend vanilla GANs by adding a desired condition to the output. In the case of handwriting generation this could mean that an output word can be conditioned to be written in regular, bold or italic style. Adding this functionality to the GANdwriter would further increase the richness of data augmentation.

4 Conclusion

This paper shows great promise in the possibility to augment handwriting datasets using GANs. However, script produced by cursive handwriting is a temporal 2D-trajectory in which ink at position (x_t, y_t) is continually connected to preceding, (x_{t-i}, y_{t-i}), and succeeding, (x_{t+i}, y_{t+i}), ink positions in time. The GANdwriter approach is not entirely able to emulate this process as the patch-implementation is a solution too general for this specific problem.

References

Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

Marius Lucian Bulacu. Statistical pattern recognition for automatic writer identification and verification. 2007.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

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.

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

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL `http://distill.pub/2016/deconv-checkerboard`.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pages 351–359, 2013.

Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.

Figure 4.1: Random examples of synthesized words compared to the ground truth. Some examples resemble the ground truth more than other examples. Most examples show that general character shapes are often approximated.

A Architecture details

The convolution and transposed convolution layers are filters with square filters of 4x4 with a stride of 2. An uneven kernel size is used because of the checkerboard problem (Odena, Dumoulin, and Olah, 2016): if the kernel size is not divisible by the stride in transposed convolution, used in the up-sampling blocks of the generator, there will be overlap in the combination of transposed convolution operations. This overlap produces pixels in the output layers that have more strength than non-overlapping pixels, as they are used in the convolution operations more frequently. An even kernel-size diminishes this effect. To manifest uniformity in the overall network architecture, all used kernels implement even-sized kernels.

All (transposed) convolutions upsample/downsample with a factor 2. Dropout layers shut down a 20% portion of their input neurons. LeakyReLU layers use a slope of 0.3.

The down-sampling blocks of the generator and discriminator consist of

`Convolution`

`Batch Normalization`

`Dropout`

`LeakyReLU`

while the up-sampling blocks of the generator consist of

`Transposed Convolution`

`Batch Normalization`

`Dropout`

`ReLU`

(adopted from (Isola et al., 2017)).

A.1 Generator architecture

Downsampling in the generator:

# **filters**	**block**
64	Conv, LeakyReLU
128	Conv, Norm, LeakyReLU
256	Conv, Norm, LeakyReLU
512	Conv, Norm, LeakyReLU
512	Conv, Norm, LeakyReLU

Upsampling in the generator:

# **filters**	**block**
512	Transp. Conv, Norm, Drop, ReLU
512	Transp. Conv, Norm, Drop, ReLU
256	Transp. Conv, Norm, ReLU
128	Transp. Conv, Norm, ReLU

After the last up-sampling layer a final layer is added which maps the number of filters to the colour channel (1 for black and white). This final layer contains a hyperbolic tangent function and has a filter size of 3x3.

A.2 Discriminator architecture

Downsampling in the 18x18 discriminator:

# **filters**	**block**
64	Conv, Norm, LeakyReLU
128	Conv, Norm, Drop, LeakyReLU

After the last down-sampling layer a final layer is added with the same purpose as in the generator architecture: to map the output to the colour-channel dimension. This final layer has a filter size of 3x3.

Discriminator architectures for other patch sizes are excluded because they differ only in the amount of layers: larger patch sizes have more layers while smaller patch sizes have fewer layers.