university of groningen

faculty of science and engineering

Towards Self-Supervised Handwritten Text Recognition Using Generative Adversarial Networks

Lisa Koopmans

university of groningen

faculty of science and engineering

University of Groningen

Towards Self-Supervised Handwritten Text Recognition Using Generative Adversarial Networks

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at University of Groningen under the supervision of
Dr. Maruf A. Dhali (Artifical Intelligence, University of Groningen)
and
Prof. dr. L.R.B. Schomaker (Artificial Intelligence, University of Groningen)
and

Lisa Koopmans (s3933083)

July 4, 2024

Abstract

Handwritten text recognition (HTR) heavily relies on supervised deep learning, which requires vast amounts of data. However, annotating handwritten documents takes much time and effort. While transfer learning and data augmentation have been used to reduce the need for labeled data, they do not eliminate it. This thesis proposes a new approach that leverages handwritten text generation for the self-supervised training of HTR models. It is based on the idea that images with the same style and text are more similar than those with different texts. In the proposed framework, synthetic images are produced using the predicted text of an HTR network. A loss function then computes the similarity between the input and synthetic images. We first explored the framework with the simpler MNIST dataset. Experiments were conducted with three different losses operating on pixels of the images or extracted feature maps. The best-performing model achieved a 91.99% classification accuracy. The findings indicate operating on higher-level features is required for decent performance. These results were used to apply the proposed framework to HTR. The generative model GANwriting was trained on half the IAM dataset to produce images with arbitrary style and content. It was integrated with a CNN-BLSTM HTR network architecture. Experiments with two image-based and two style-invariant losses were conducted, where self-supervised HTR models were trained on data independent from GANwriting. When no useful information was learned by the models, stylistic differences between input and synthetic images were minimized by conducting experiments with the image-based losses on data recreated with GANwriting. The results on recreated data showed the image-based self-supervised models could only achieve good performances when using transfer learning. Starting from a pretrained HTR model, the best-performing model had character and word error rates of 0.44% and 1.56% on synthetic test data, and 36.08% and 66.40% on real test data. Overall, our findings showed a high-quality generative model and background knowledge are necessary for adequate performance of image-based self-supervised HTR.

Acknowledgments

This thesis has been a significant challenge, teaching me how to approach research when the results are not as promising. I would like to express my gratitude to Dr. Maruf Dhali for his invaluable ideas and guidance that shaped the experiments of this research and kept me motivated. I would also like to thank Prof. Dr. Lambert Schomaker for his insights that enabled me to approach the problem from various angles and for inspiring me to tackle the complexities of handwritten text recognition with self-supervised learning. Lastly, I thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high performance computing cluster.

Contents

1 Introduction

Machine learning algorithms are commonly divided in two categories: supervised learning and unsupervised learning. In supervised learning, an algorithm learns a mapping from the input data to an output using a set of corresponding labels, whereas in unsupervised learning, such labels are not given. Unsupervised learning algorithms instead rely on the similarity between data samples to learn about the properties of the underlying structure and relationships in the data to produce an answer [1]. Many problems are solved as supervised problems, specifically so in deep learning, where neural networks are trained by minimizing the error between the network prediction and the ground truth label. One field in particular that heavily relies on supervised learning is handwritten text recognition (HTR), which is the task of transcribing scanned handwritten documents to computer-readable text (e.g. ASCII). This allows for further applications on the scanned documents such as keyword searches and machine translation.

A good HTR system is able to transcribe handwriting in any writing style and text. This results in a complex task with highly variable data. Considering the writing styles, for example, HTR models need to deal with the between- and within-writer variability; each person has their own writing style, and a single person's handwriting varies each time they write. This is caused by both biological differences and environmental influences, which affect, for example, the pen grip, physical position, and pen pressure. On paper, this variability is visible through differences in, e.g., slant, curvature, and character size. The task of differentiating between writer styles, also known as writer identification, has been studied for decades, which further shows the complexity of the task.

As HTR is the task of transcribing handwritten texts from images to digital text, it involves language processing. The endless number of character combinations for the formation of words and sentences significantly contributes to the HTR problem's complexity. HTR is commonly dealt with as a classification or sequence processing problem. A classification approach is simpler since it does not have to account for time dependencies, but when applied to higher levels than singular characters, it quickly becomes infeasible. Already at the word level, there are thousands of possible classes. Sequence processing approaches predict the likelihood of a character being present at a time step, which allows for greater flexibility and modeling of character dependencies. However, these HTR models learn the dependencies specific to the language and lexicon present in the data they have been trained on, causing them to struggle with generalizing to unseen textual contents or capturing rare character combinations. Additionally, both classification and sequence processing approaches would not be able to deal with texts written in scripts not present in the data.

Due to the highly variable nature of the data for HTR tasks w.r.t. writer styles, languages, and scripts, HTR systems require a large number of annotated samples to produce accurate transcriptions and generalize well. However, obtaining these takes much time and effort. This is especially the case for the application to handwritten documents, where expertise in paleography is required. Previous research has explored various methods to improve HTR performance when working with a limited number and variety of annotated data. Two common methods are transfer learning and data augmentation. Transfer learning encompasses a knowledge transfer between neural networks by fine-tuning the weights of a neural network that is pretrained on a related task. This can speed up training times and has been shown to improve HTR performance [2]. With data augmentation, the data variability is increased by creating new samples through the application of, e.g., geometrical and morphological operations to existing data. This can also be done with handwritten text generation (HTG), a field that aims to produce realistic images of handwriting in arbitrary writer styles and text. Synthetic images produced with these methods have been shown to improve the quality of transcriptions for both modern and historical documents [3], [4].

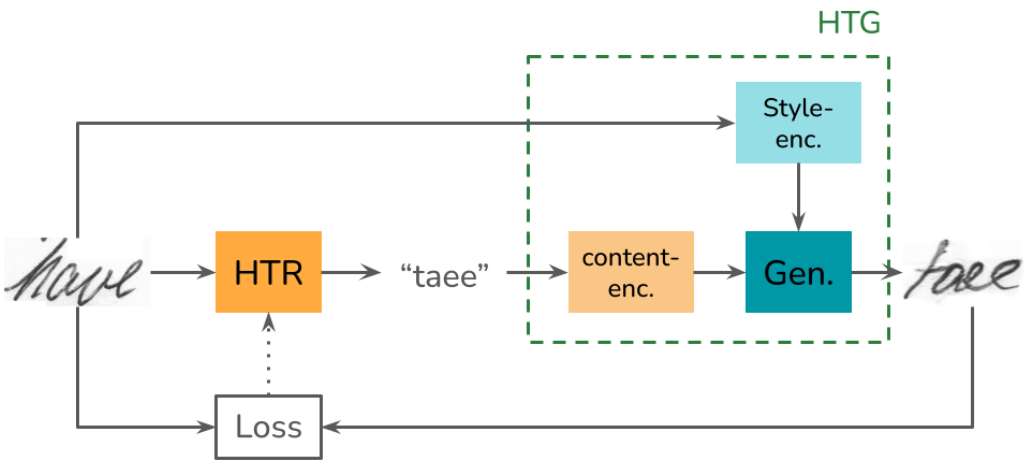


Figure 1: The proposed framework for image-based self-supervised HTR. It uses a pretrained HTG conditioned on style and textual content representations, resulting from their respective encoders, to generate synthetic images with HTR-predicted text. A loss function computing the dissimilarity between the input and synthetic images then informs the HTR of the quality of its prediction.

Producing images with arbitrary writer styles and text, HTG has the potential to eliminate the need of real data for HTR systems. However, despite the rapid advances in HTG in recent years, synthetic images are not yet representative enough of real handwritten samples and thus cannot yet replace real data [5]. The approaches to data augmentation that apply transformations to existing data are further limited in that they cannot increase the data's variability w.r.t. language. Moreover, transfer learning approaches still require annotated data for fine-tuning the pretrained networks to the HTR task. Hence, there remains the question of whether the need of annotated data to train HTR systems can be further reduced and even eliminated. This is the matter that the current thesis aims to address. To do so, we consider a type of unsupervised learning that does not rely on labels, but where the data serves as its own ground truth: self-supervised learning.

Krishnan et al. [6] proposed TextStyleBrush, which uses a partially self-supervised approach to text generation for style and content disentanglement based on the similarity of real and generated images. The idea behind the self-supervision of TextStyleBrush is that texts in the same font or writing style are more similar compared to texts in differing styles. Likewise, it could be argued that two images in the same writer style, and with the same texts are more similar to each other than two images in the same style, but with differing texts. Inspired by TextStyleBrush, this thesis hence aims to explore the notion of self-supervised learning through image-based similarity to eliminate the need for labels when training an HTR system. Specifically, we propose a novel self-supervised framework in which an HTR system is trained based on the similarity of an input image and a synthetic image that contains the HTR-predicted text. Pretrained HTG models are ideal for producing these synthetic images, as they can produce images with arbitrary texts and ensure the writer styles in the input and synthetic images are similar. Training an HTR system in such a manner also allows for further improvement of existing models on unseen, unlabelled data and, therefore, could be an alternative to post-processing methods that leverage language models.

The schematic of the proposed framework in Figure 1 portrays how the pretrained generator of an HTG model can be integrated with HTR for self-supervised training. It is essentially an extension

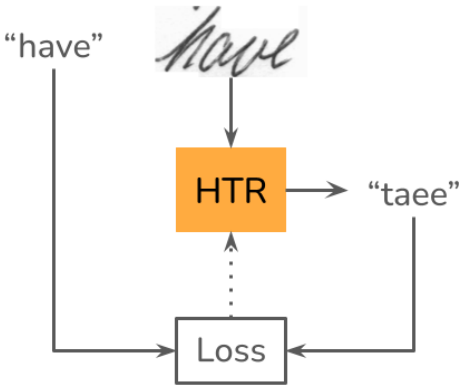


Figure 2: The standard supervised HTR framework. Here, a loss function informs the HTR network of the quality of its prediction through a comparison with the ground truth label that corresponds to the input image.

of the supervised HTR framework as shown in Figure 2, where instead of the text label, the input image is considered the ground truth, and the generator of an HTG model is inserted between the HTR network prediction and loss computation. The question remains, however, how the integration of HTG and HTR models can be implemented. Moreover, unlike supervised HTR, the proposed framework should learn based on image similarity. Due to the novelty of the framework, it is an open question of what type of loss function is appropriate for learning differences in handwritten texts and images. The aforementioned questions as well as the feasibility of the framework were first explored with handwritten character recognition, being the simplest approach to HTR as it is an image classification task. Then, the proposed image-based self-supervised framework was investigated with the more complex sequence processing task of handwritten word recognition.

For the implementation of the proposed framework, it has to be noted that HTG models are conditional generative networks. Typically, a style encoder extracts a latent representation of the desired writer style based on single or multiple images. Similarly, a content encoder extracts a latent representation of the desired text. These style and content representations are then input into a generator network to produce a synthetic image. The general process of this is portrayed in Figure 1. Adapting the supervised HTR framework to a self-supervised framework then requires the generator to take in the HTR-predicted text and encoded style of the input image to produce a synthetic image. Hence, the HTR predictions need to be mapped to the latent content representation. Training any neural network, however, requires that all operations performed on the input that lead to the loss computation are differentiable. Therefore, we propose to directly input the predicted class probabilities of the HTR model to the generator. To enable this, the generator is conditioned on one-hot encoded text labels.

Neural networks are trained by minimizing the objective function, i.e., the loss function. Therefore, an appropriate loss function is necessary for a network to learn information relevant to the task. The proposed framework aims to maximize the similarity, i.e. minimize the dissimilarity, between real and synthetic images based on their textual content. Therefore, an appropriate loss function for image-based self-supervised HTR needs to capture the differences in text such that the correct characters are predicted and present in the synthetic data. Taking inspiration from image reconstruction and generation tasks, we experimented with three image-based losses that operate directly on the image pixels or on higher-level information by comparing extracted feature maps. Results suggested that a higher level of information is necessary for substantial learning to occur in the image-based self-

supervised framework. This was used to apply image-based self-supervised learning with HTG on word recognition. When initial experiments with these image-based losses did not show promising results, inspiration was also drawn from the field of word spotting, where images are matched based on textual content regardless of style. As such, two style-invariant losses were investigated in addition to the image-based losses.

1.1 Research Questions

To summarize, this thesis aims to reduce the need of annotated data for the training of HTR systems through image-based self-supervised learning and HTG. In order to be effectively used for HTR, such a system should be able to learn knowledge of the task by itself, as well as further improve existing HTR models on new data. However, due to its novelty, it has to be explored how to implement and train such a framework. As such, the following primary and secondary research questions are studied:

Can self-supervised learning based on the similarity between real images and synthetic images that contain predicted text and are produced with handwritten text generation models be effectively used for handwritten text recognition?

RQ1 How can handwritten text generation be effectively incorporated into image-based self-supervised handwritten text recognition?

RQ2 What is an appropriate loss function for training the image-based self-supervised handwritten text recognition framework?

RQ3 Can pretrained supervised handwritten text recognition models be improved using image-based self-supervised learning?

1.2 Thesis Outline

First, Chapter 2 provides the relevant theoretical background and related literature on HTR and HTG. In Chapter 3, the primary research question as well as the first and second secondary questions are addressed through the application of the proposed image-based self-supervised framework to handwritten character recognition. Based on these findings, the methodology to answer the primary and secondary research questions for handwritten word recognition is explained in Chapter 4. The results are then presented in Chapter 5, for which conclusions are made concerning the research questions in Chapter 6. Lastly, Chapters 7 discusses the weaknesses of the proposed methodology and suggests directions for future research.

2 Background

2.1 A Brief Overview of Training Neural Networks

This section serves to provide a brief explanation of how neural networks are trained. Additionally, we explain the types that we focus on in this thesis. The explanations provided in this section are based on [1].

Neural networks can be described as non-linear function approximators that use layers of nodes, also termed neurons, that have weighted connections to each other. The input and output layers are considered visible layers, and any layer in between are hidden layers. A very basic neural network with multiple layers is the multi-layer perception (MLP). Deep neural networks can then be described as variations of MLPs with over three hidden layers. They are hierarchical models, in which each hidden layer learns features on a lower level than the layer before it, allowing it to form complex representations. This is possible due to the non-linearity that activation functions introduce. The activation function determines the output of a neuron. How the output $\mathbf{h}$ of a layer with weights $\mathbf{W}$ is computed for an input $\mathbf{x}$ can be described mathematically as in Equation 1. The activation function is indicated by σ, and the vector $\mathbf{b}$ is the bias term, which are trainable parameters that shift the activation function.

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b}) \tag{1}$$

Now that it is clear what a neural network consists of, the question remains how it exactly learns to approximate functions. Neural networks are trained by iteratively updating the weights such that the error between the network predictions and what the network should have predicted is minimized. This error is computed by a cost function, also called the loss function. The network weights are updated by moving in the direction of the negative gradient of the loss function, a process called gradient descent. The magnitude of the updates is controlled by the learning rate. By basing the network weight updates on the loss's gradient, the possible loss functions are constrained to include only those that are differentiable. Since computing the gradient of the loss over the entire dataset is computationally expensive, the gradient is commonly evaluated over a randomly sampled single data point (stochastic gradient descent), or over a randomly sampled mini-batch. Different variations of stochastic gradient descent optimization have been proposed for more stable and faster training by, e.g., including past gradients in the calculations.

To minimize training times, it is important that the gradient computation can be done as efficiently as possible. For this, the back-propagation algorithm is used. Given a neural network with i layers and parameters (weights and biases) θ_i for each function $\mathbf{h}_i$, the gradient of the loss is defined as the partial derivatives of L with respect to each θ_i, i.e. $\nabla L = [\frac{\delta L}{\delta \theta_1}(\theta), ..., \frac{\delta L}{\delta \theta_i}(\theta)]$. The back-propagation algorithm computes the partial derivatives efficiently by recursively applying the chain rule to the input of each function $\mathbf{h}_i$. If any further operations are performed on the network output before the loss computation, the partial derivatives over these operations are also computed. In libraries specialized for training neural networks such as TensorFlow and PyTorch, each operation involved in the loss computation is traced and automatically differentiated. The gradients are stored during the backward pass.

A model that can only perform well for the specific cases present in the training data has not learned about the data generating distribution (also source distribution). Therefore, neural networks are tested on unseen data to ensure they have not only learned the specific patterns in the training data but that they generalize well. Next to training and test sets, a validation set is used for tuning a network's hyper-parameters. If a model has a high loss on the training set, it indicates it cannot

capture well the patterns in the data; the model is underfitting. On the other hand, when the difference between the training loss and validation loss is increasing or large, the model is too specific to the training set and is overfitting. To prevent this, a network can be regularized by for example applying additional constraints to the loss function.

2.1.1 Convolutional and Recurrent Neural Networks

In this thesis, we will use two types of neural networks: feed-forward neural networks, specifically convolutional neural networks (CNN), and recurrent neural networks (RNN). In feed-forward networks, the information is propagated forward through the network layers without any feedback cycles. The CNN is a specialized type of MLP where convolutions are applied to grid-like data (commonly image data). A convolutional layer consists of multiple kernels that move over the input and are convolved with it at each time step. Using the same kernel multiple times on the same input (i.e., weight sharing) allows for a significant reduction in parameters compared to dense layers, where each neuron in one layer is connected to each neuron in the next. To maintain the data's dimensionality, padding is usually applied. Additionally, pooling operations further reduce the number of parameters and add invariance to small translations by summarizing neighborhoods of the data. Consecutively applying convolutions to an input image allows for hierarchical feature extraction, as each deeper layer extracts more abstract features.

RNNs are a type of neural network specific to sequential data. In RNNs, the output at the current time step depends on the input at the current time step and the output of the previous time step. For this, an RNN unit has a hidden state that captures the sequence information, serving as a 'memory'. RNNs notoriously suffer from the vanishing and exploding gradients problems, which occur when the gradients become near zero (vanishing), or very large (exploding). This is because the gradients of RNNs are computed with back-propagation through time, where the back-propagation algorithm is applied to the unfolded feedback cycles in the recurrent units. The unfolded RNN is usually very deep and consists of repeated operations on the same weights as network weights are shared across time steps, making the vanishing and exploding gradients problems much more likely to occur. Another problem with RNNs is that the output at each time step is most influenced by the most recent output of an earlier time step (i.e., $t-1$), which makes long-term dependencies difficult to obtain. One way to mitigate this problem is to introduce connections with a higher time delay. This also reduces the vanishing and exploding gradients problems, as the gradients vanish or explode exponentially with respect to the number of time-steps [7].

2.2 Handwritten Text Recognition

Handwritten text recognition is a challenging problem that researchers have been tackling for decades. This is because the data does not originate from the same source distribution; there is a large variety in handwriting styles and language structures produced by different people. In other words, the data is not independently, identically distributed.

Except at the character level, HTR is considered a sequential problem. Before deep learning became feasible, hidden Markov models (HMM) were employed where word models were built based on character-level features in implicit- and explicit segmentation methods [8]. Later, hybrid approaches that combine neural networks with HMMs were proposed for line recognition [9]. However, these models were limited as HMMs do not consider context and need hand-crafted features for a good performance. Due to their ability to incorporate context and sequential nature, RNNs were a viable alternative to HMMs, and have become a key component in HTR models. RNNs output

a probability distribution over the possible characters for each time step. To train RNNs for HTR, Graves et al. [10] proposed the CTC framework, where the inclusion of a blank, no-label, symbol eliminated the need for explicit alignments between the images and labels.

Standard RNNs are prone to suffer from the vanishing and exploding gradients problems, and struggle to maintain long-term dependencies. To mitigate these issues, long short-term memory (LSTM) architectures, a type of RNN, are used in HTR systems with the CTC framework [11]. Inside an LSTM unit, three gates, each with their own learnable parameters, control the information flow based on the current input and the output at the previous time step. Additionally, a cell state stores the cross-time dependencies. A forget gate controls what of the previous cell state to use, an input gate controls what of the input at the current time step to use, and an output gate controls what information of the current cell state to output [12]. Standard, unidirectional LSTMs are still limited, however, as they only consider past information. Hence, bi-directional LSTMs (BLSTMs), where two LSTM layers simultaneously process the input sequence in forward and backward directions, are commonly used instead.

Though combining BLSTMs and the CTC framework improved the state-of-the-art at the time, it did not eliminate the need for hand-crafted features. Moreover, it was required to reduce stylistic differences between samples in preprocessing for good performances [11]. Circumventing the need for hand-crafted features, Graves and Schmidhuber [13], input raw pixels to a multidimensional LSTM (MDLSTM) network with CTC, which adds recurrent connections along both the spatial and temporal dimensions of the data. Later, Shi et al. [14], proposed a convolutional recurrent neural network (CRNN) architecture, where features are extracted with a CNN and an RNN predicts the label sequences. Applying MDLSTMs to a variation of this architecture, Voigtlaender et al. [15] outperformed the previous MDLSTM frameworks. While showing impressive results, MDLSTMs are computationally expensive. Consequently, research turned to cheaper alternatives.

Puigcerver [16] showed that using a CNN with BLSTM layers could achieve similar performance to CNN-MDLSTMs while being computationally less expensive. Alternatively, Bluche and Messina [17] proposed the Gated-CRNN (GRCNN), where gates in the CNN determined a feature's relevancy for a certain position. In [18], de Sousa Neto et al. aimed to improve upon Puigcerver [16]'s results, while maintaining a low number of parameters like Bluche and Messina [17]. They adapted the GRCNN such that the gates were applied to only half the features. Additionally, they used bi-directional gated recurrent units (BGRU) instead of BLSTMs. A BGRU unit has a hidden state and two gates: a reset gate that allows for discarding information irrelevant to the future, and an update gate that controls the information flow from the previous hidden state [19]. The proposed architecture outperformed both Puigcerver [16] and Bluche and Messina [17].

Alternative approaches to RNN-based HTR have also been investigated, specifically on the word level. Almazán et al. [20] simplified word spotting and recognition to the nearest neighbor search by mapping the text labels and features extracted from word images to a common d-dimensional character-level word attribute space, termed pyramid of histogram characters (PHOC). Krishnan et al. [21] built upon this by using deep CNN features of a word image classification model, HWNet [22], instead of hand-crafted ones. Recently, an improved version of HWNet was applied to an adapted version of the PHOC framework, where a common subspace for a joint feature representation between text labels and word images was learned, reaching state-of-the-art results [23]. These nearest-neighbor approaches are limited, however, as they rely on lexicons. Avoiding this, Mondal et al. [24] applied object recognition for sequential character detection and identification to recognize English text.

Like how nearest-neighbour approaches rely on lexicons, many of the RNN-based architectures rely on statistical language models in post-processing to obtain improved results. This can be especially useful for out-of-vocabulary (OOV) words, which HTR models struggle to generalize to. These

language models are typically n-gram language models trained on an external corpus or on the lexicon of the training data [13], [15] - [18]. N-gram language models give a probability of the next word or character based on the previous $n-1$ words or characters. Besides language models, heuristics are also used for decoding the RNN predictions. One such method that is used at inference time to improve recognition results is the beam search algorithm [8], [18], which performs a tree-search where only the n-best candidate sequences at each time step are considered [25]. Beam search can operate on both the character and word level, and incorporate language models [26].

In recent years, HTR research has focused more on attention-based architectures. These models reduce the need for language models in post-processing by considering the language modeling abilities of the networks themselves. Attention-based models are commonly sequence-to-sequence (seq2seq) encoder-decoder models, where an encoder produces a feature sequence, and the decoder maps this to characters. An attention mechanism then allows for better alignment between input images and their labels by focusing on more relevant features at each decoding step. Michael et al. [27] proposed such a network, achieving competitive results using only beam search. Around the same time, Kang et al. [28] proposed a seq2seq model architecture, but with content- and location-based attention as well as label smoothing. Additionally, Poulos and Valle [29] explored non-linear attention mechanisms. Their model was trained s.t. the current character prediction was conditioned on the previous one, which would enable better generalization to out-of-vocabulary words.

Following the successes of the transformer architecture in natural language processing, HTR models started to adopt this architecture as well. Transformers [30] address the memory limitations of recurrent networks for longer sequences and their lack of parallelization capabilities by relying on attention mechanisms. Kang et al. [31] proposed a transformer-based architecture that operates on the character level and can learn language-related character dependencies. This allows for generalization to OOV words without the use of explicit language models. While Kang et al. [31] used a ResNet first to extract one-dimensional representations of text lines, Li et al. [32] used a pretrained vision transformer and initialized the decoder weights with a pretrained language model in TrOCR. The latter, they motivate, altogether eliminates the need for language models in post-processing. Leveraging these models proved to result in high performance, as the authors report they achieved new state-of-the-art results. The downside of transformer architectures, however, is that they are computationally expensive and require especially large amounts of labeled data. For comparison, the CNN-BGRU HTR-Flor model [18] had 0.8 million parameters, whereas the smallest TrOCR model [32] had 64 million parameters.

2.3 Handwritten Text Generation

HTG aims to produce realistic images of handwritten text, written in a desired style, and with a desired text. The main motivation behind HTG is to leverage it for HTR, where data labels are costly to obtain. Especially the ability to produce images conditioned on content would enable a greater variety of words present in the data, potentially aiding in the prediction of OOV words.

Early works on HTG concatenate handwritten glyphs consisting of one to three characters through fitted curves or polynomials [33]. Another approach, by Graves [34], was to use LSTMs with a sliding window over a desired character sequence for synthesizing text line images. However, since Alonso et al. [35] proposed the use of generative adversarial networks (GANs) for the generation of handwritten word images, GANs have been the focus of HTG. A standard GAN consists of two models: a generator G that produces an image from randomly sampled noise $\mathbf{z}$, and a discriminator D that discriminates between synthetic and real images. The objective of the generator is to produce images that the discriminator deems as real, while that of the discriminator is to correctly identify these

as synthetic images. This can be summarized in the objective function given in Equation 2, where $p_{data}(\mathbf{x})$ is the data distribution, $\mathbf{x}$ the data, and $p_z(\mathbf{z})$ the prior of the noise. The first term represents the discriminator's objective function, while the second term represents that of the generator. For more details on the GAN, we refer to [36].

$$\min_G \max_D L(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log(1-D(G(\mathbf{z})))] \tag{2}$$

Alonso et al. [35] adapted the GAN in two ways. First, they conditioned the generator on content by concatenating a content embedding encoded with a BLSTM network with the noise input and injecting it into intermediate layers of the generator network. Second, inspired by Odena et al. [37], a text recognition network was added to enforce the content in the generated images. This HTR model was trained concurrently with the generator and discriminator models with the CTC loss. To incorporate it in the GAN, its loss on the synthetic images, and the conditioned content input was added as an extra term to the generator's objective function. While the HTG model was able to generate images with a certain content, the images did not look natural; artifacts were present, and ink traces were blurry. Additionally, the style could not be controlled.

Fogel et al. [38] introduced ScrabbleGAN, which could produce images of unconstrained lengths with arbitrary contents. The network could do this by generating each character separately and leveraging the receptive fields of CNNs to learn connections between characters. The discriminator worked similarly. As in [35], ScrabbleGAN used an auxiliary text recognizer. Additionally, the style was controlled with noise vectors that were kept constant between characters of the same word. Both Alonso et al. [35] and Fogel et al. [38] added 100,000 synthetic images to two separate HTR benchmark datasets and showed a slight improvement in performance. However, considering the large amount of data samples added, their effectiveness w.r.t. generalization to real data is questionable.

While ScrabbleGAN could generate images with characters in the same style based on a noise vector, it could not reproduce the styles of specified writers. Kang et al. [39] noted the model suffers from mode collapse due to its low variability in writing styles, which can be partly attributed to the assumption that all characters have the same widths. Conditioned on both style and content, GANwriting [39] generated more realistic images than previous HTG models. To condition the GAN on style, the authors extracted a style embedding in a few-shot manner from word images of the same writer. The content was embedded both character-wise and globally to allow for OOV word generation. To enforce the conditioned style and content in the synthesized images, losses from an auxiliary writer classifier and HTR network applied to synthetic data were added to the generator loss as separate terms. The synthetic word images appeared more realistic and readable compared to previous models, but GANwriting has a prominent weakness: it can only produce good quality word images when the words are no longer than seven characters.

HiGAN [40] uses a similar approach to ScrabbleGAN for content encoding, enabling it to produce images of arbitrary lengths. It overcomes the need for few-shot style extraction by introducing a style reconstruction loss that minimizes the difference in encoded styles of the real and synthetic images. Additionally, a writer classifier was used, as well as a regularization term to match the style latent space with the prior random distribution. The authors improved HiGAN in [41], terming their new model HiGAN+. They reduced artifacts such as blurriness and improved style imitation by adding a patch-level discriminator, a content reconstruction loss, and a contextual loss between high-level features extracted with the writer classifier. While HiGAN+ produced higher-quality images than previous HTG models, it is a complex model with eight loss terms for the generator's objective function, which shows that HTG is a complex problem.

Following a similar paradigm with auxiliary networks, several other GAN-based models have

been proposed. For example, TextStyleBrush [6] was trained with a self-supervised framework combining a text-specific perceptual loss, two reconstruction losses that compare high-level features between synthetic and input images, and a content loss. While TextStyleBrush has high potential due to its self-supervised nature, its implementation details were not reported in detail. Another model, JokerGAN [42], enhanced the quality of synthetic images by adding an input specifying the vertical position of characters. Finally, AFFGANwriting [43] improved image quality by fusing local and global style features extracted with a VGG-19-based style encoder.

Like HTR, research on HTG has also incorporated transformer architectures in their frameworks. For example, Kang et al. [44] extended GANwriting to the line level and replaced the seq2seq HTR model with a transformer architecture. Moreover, the authors of JokerGAN incorporated a vision transformer for character-wise content encoding in their adapted architecture, JokerGAN++ [45]. Bhunia et al. [46] claim to be the first to propose a transformer-based generative network, called Handwriting Transformers (HWT), where the self-attention mechanisms allow for the encoding of both global and local style features. Building upon HWT, Pippi et al. [47] improved the quality of synthetic images for rare characters by encoding the desired text as a concatenation of vector representations of its corresponding unifont character images.

More recently, diffusion models have gained more attention in HTG research due to their high image quality. Diffusion models are trained by learning to reverse a process where sampled noise is added to input data across time steps such that the data ends up looking like Gaussian noise [48]. HTG literature commonly uses the Denoising Diffusion Probabilistic Model (DDPM) [49], which uses an alternative loss function based on the error between the true noise and the estimated noise. One such method proposed for HTG is Conditional Text Image Generation with Diffusion Models (CTIG-DM) [50]. The network is conditioned on encodings of the unique visual characteristics of the input images, semantic context, and writer style by inputting them to the network at each time step with a corresponding input image. The authors showed that an HTR model trained on 200,000 synthetic images generated by the CTIG-DM network was able to achieve model performance similar to the HTR model trained on only real data, and a significantly better performance when trained on mixed data.

DDPMs are lengthy to train and require high computational resources as they operate directly on images and a separate model is trained at each time step. Hence, latent diffusion models (LDM) were proposed [51], where the diffusion process is applied to an image's latent representation. This method was used for WordStylist [5], where an embedding layer and a transformer block were used to encode the desired writer index and content, respectively. The style encoding and noise vector were directly input to the noise predictor network, and the content encoding was injected into intermediate layers of the network. The authors compared WordStylist with two other models: GANwriting [39], and SmartPatch [52], the latter of which extends GANwriting with a patch-based discriminator and HTR-based attention mask. They showed an improved performance of WordStylist, both qualitatively and quantitatively. One method they used to show this is by training a word-level HTR model on the IAM database separately with a training set of real images and with the same training set recreated with the different HTG models. The HTR models were then tested on real data for comparison. To then give an indication of the effectiveness of HTG models for HTR, we show the results for this method reported by Nikolaidou et al. [5] in Table 1, where the CER and WER are the character and word error rates, respectively.

Table 1: Test results of an HTR model tested on real IAM data. The HTR models were trained on real training IAM data, and on the training data recreated with GANwriting, SmartPatch, and WordStylist. The results are as reported in [5].

Training Data	CER (%)	WER (%)
Real IAM	4.86 ± 0.07	14.11 ± 0.12
GANwriting IAM	38.74 ± 0.57	68.47 ± 0.32
SmartPatch IAM	36.63 ± 0.71	65.25 ± 1.02
WordStylist IAM	8.80 ± 0.12	21.93 ± 0.17
Real IAM + GANwriting IAM	4.87 ± 0.09	13.88 ± 0.10
Real IAM + SmartPatch IAM	4.83 ± 0.08	13.90 ± 0.22
Real IAM + WordStylist IAM	$\mathbf{4.67 \pm 0.08}$	$\mathbf{13.28 \pm 0.20}$

3 Testing the Framework: MNIST

This Chapter explores the feasibility of the proposed image-based self-supervised learning framework by applying it to Handwritten Character Recognition (HCR). Being a classification problem of single characters, this minimizes the complexity of the task compared to higher-level HTR. Specifically, the framework was applied to the handwritten single digits of the benchmark dataset MNIST. Simplifying the HTR problem in this manner allows us to focus on 1) whether the proposed framework has the potential to be used for word-level HTR, 2) how the proposed framework can be implemented, and 3) what loss functions could be used for self-supervised HTR. We first show that the character recognition model can achieve a good performance in a supervised setting before integrating it with a pretrained generator into the self-supervised framework.

The experiments detailed in this Chapter were implemented in Python. The models were implemented with TensorFlow [53]. The code is publicly available at `https://github.com/Lisa-dk/self-supervised-mnist.git`.

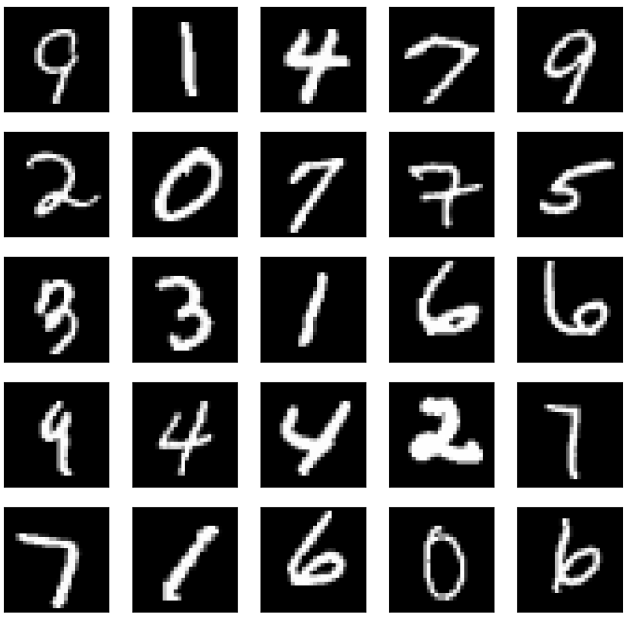


Figure 3: Example images from the MNIST dataset.

3.1 Dataset

To test whether the proposed framework can be used for HCR, we used the MNIST dataset [54]. This is a widely used benchmark dataset for handwritten digit classification on which relatively simple neural networks achieve high performances. This is due to the fact that the images contain little noise, and the digits are centered in each image. Consequently, algorithms only have to account for the differences between the character shapes and not for their spatial positions. The MNIST dataset consists of 70,000 28 x 28 grayscale images of the handwritten digits 0 to 9. The images have pixel values between 0 and 255. Examples are shown in Figure 3. Figure 4 shows that the classes of the dataset are uniformly distributed. Moreover, to get an indication of class separability, we visualized the data with t-SNE on the flattened raw images in Figure 5. Overall, it shows clear clusters for each digit class, indicating that they are separable.

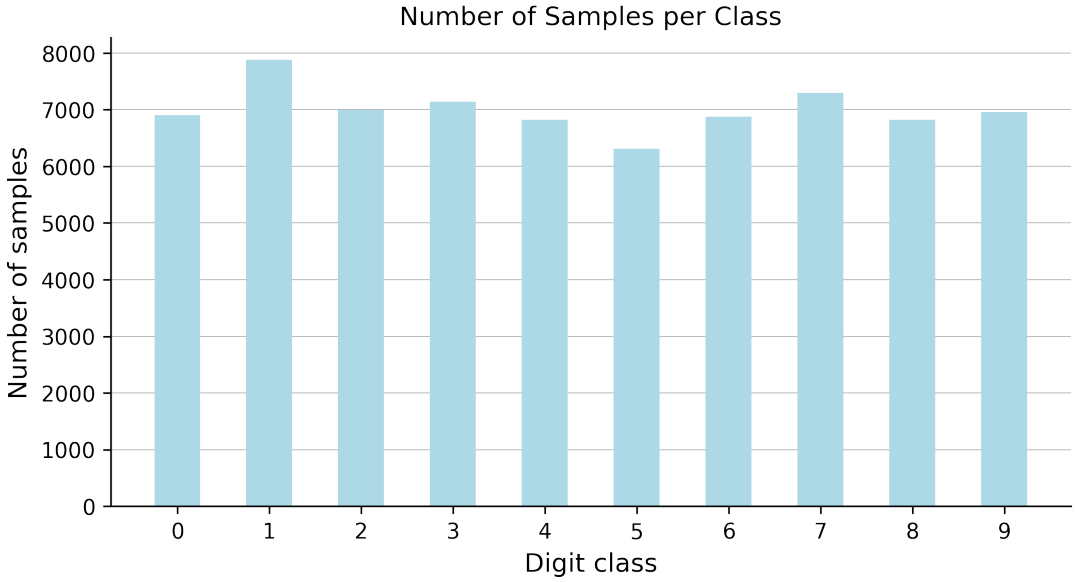


Figure 4: Number of samples per class of the MNIST dataset

We used the training and testing splits constructed by Lecun et al. [54], containing 60,000 and 10,000 images, respectively. To maintain independence between the character recognition and character generation models, the training set was split in half, resulting in two datasets: MNIST-HCR, and MNIST-GEN. The test set served as a hold-out set for MNIST-HCR. Consequently, there are 30,000 training images for the MNIST-GEN and MNIST-HCR datasets each, and 10,000 test images for MNIST-HCR. All images were divided by 255 to rescale them in the range of [0, 1]. This preprocessing step was performed for all images in both datasets to ensure compatibility between the HCR and generative models.

3.2 Supervised Handwritten Character Recognition

3.2.1 Model Architecture

Keeping the integration of the HCR architecture with the pretrained HCG model into the self-supervised framework in mind, we prioritized simplicity over an optimal supervised model performance. Hence, we designed a simple three-layer CNN network for HCR that, although not state-of-the-art, could achieve a good performance. As shown in Figure 6, our CNN architecture consists of three convolutional layers for feature extraction, where the first two layers have 64 channels, and the third 128 channels. To each layer, 3 x 3 kernels were applied. In the first two layers, this was with stride 2, and in the third layer, with stride 1. The activation function was Leaky ReLU, which has a small gradient for negative inputs, preventing a discontinued gradient flow that occurs otherwise with the standard ReLU function [1]. After the first two convolutional layers dropout was applied with a probability of 0.2 to prevent overfitting. The network then had a final 10-dimensional dense layer, where the Softmax activation function computed the class probabilities for each of the 10 digits.

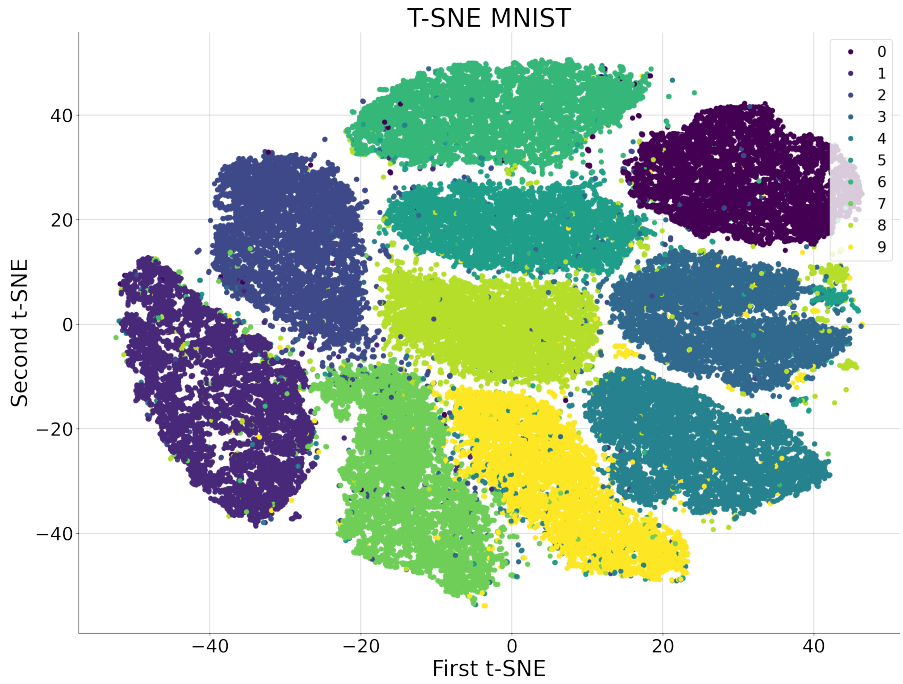


Figure 5: Data visualization of the MNIST dataset with t-SNE.

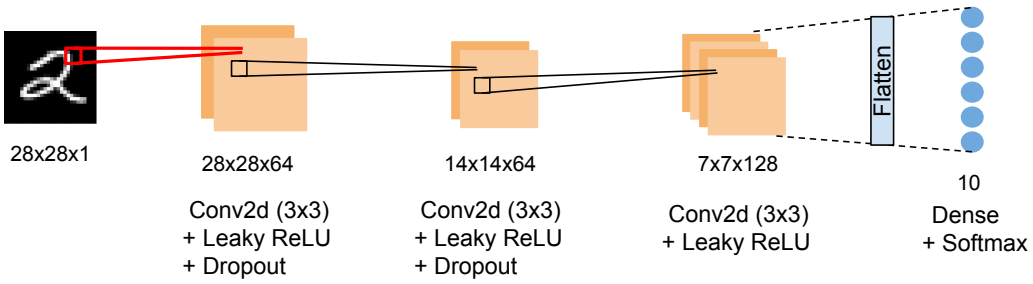


Figure 6: The CNN architecture for supervised and self-supervised HCR, using three convolutional layers for feature extraction and a dense layer for digit classification.

3.2.2 Implementation Details

Since character recognition is a multiclass classification problem, the HCR architecture was trained with the Cross-Entropy loss as expressed in Equation 3. Here, C is the number of classes, y_i is the probability of the ground truth label for class i, and $\hat{y}_i$ is the predicted probability for class i. Moreover, the Adam optimizer was applied with a learning rate of 0.001 and the batch size was set to 256. The learning rate and batch size were chosen as they resulted in good performance but were not further optimized with hyperparameter tuning.

$$L_{CE} = -\sum_{i=1}^{C} y_i log(\hat{y}_i) \tag{3}$$

3.2.3 Evaluation

The supervised HCR model was trained with 5-fold cross-validation on the MNIST-HCR dataset. To prevent overfitting, early stopping was applied to the validation loss with a patience of 5 epochs. To minimize the dependency of the results on the weight initialization and training set, the test results were obtained by taking the mean of the predicted class probabilities per k-fold. Since there were no heavy class imbalances, the models were evaluated with accuracy, i.e., the fraction of correctly predicted samples. In addition, the confusion matrix was computed on the test set to gain better insight into the model's performance. To contextualize the model's performance, the final test results were compared with DropConnect, a state-of-the-art model by Wan et al. [55]. Here, the model used dropout not on the activations, but on the network weights. They used data augmentation and a learning rate scheduler as well as a smaller batch size of 128. Moreover, Wan et al. [55] obtained their test results in the same manner as we did. Because the same test set was used, the results could be directly compared.

3.3 Handwritten Character Generation

3.3.1 Model Architecture

A generative architecture that has been shown to perform well for character generation with the MNIST dataset is the Deep Convolutional GAN (DCGAN) [56]. This architecture uses convolutions for feature extraction in the discriminator and up-sampling in the generator, which allows for a higher resolution of synthetic images and more stable training. Moreover, a DCGAN is more simple than other proposed variations of character generators and will thus be easier to integrate with the image-based self-supervised framework. It should, however, be conditioned on the digit classes. One such model was implemented by Keras [57] and is publicly available[1]. The model is a DCGAN where both the discriminator and generator are conditioned on the digit classes, following the approach of Mirza and Osindero [58]. We adapted this model to TensorFlow to ensure compatibility with the HCR architecture.

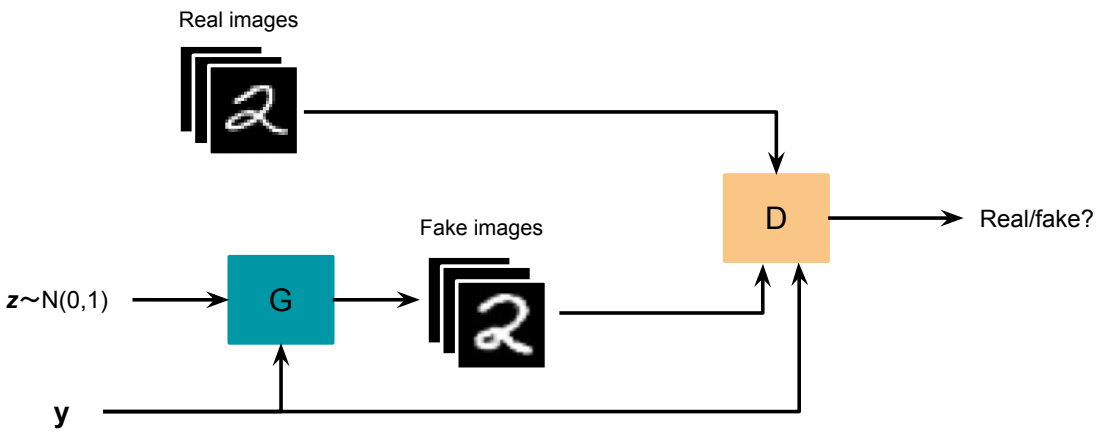


Figure 7: Overview of Chollet et al. [57]'s conditional DCGAN architecture. Here, a generator G produces synthetic images from randomly sampled noise $\mathbf{z}$ with the given digits $\mathbf{y}$. A discriminator predicts whether the images of the given digits are real or not.

[1] https://github.com/keras-team/keras-io/blob/master/examples/generative/conditional_gan.py

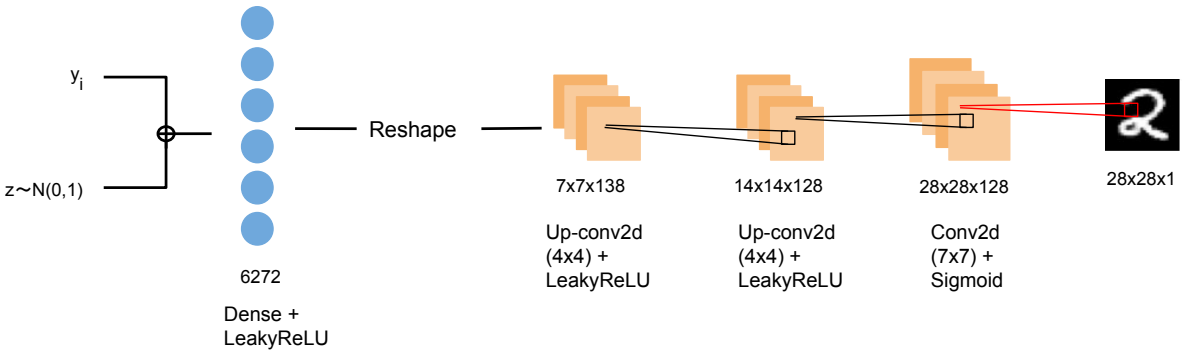


Figure 8: The DCGAN's generator architecture, using one dense and three up-sampling layers to produce synthetic images. It was conditioned on the digit classes by embedding the randomly sampled noise z with a label y_i.

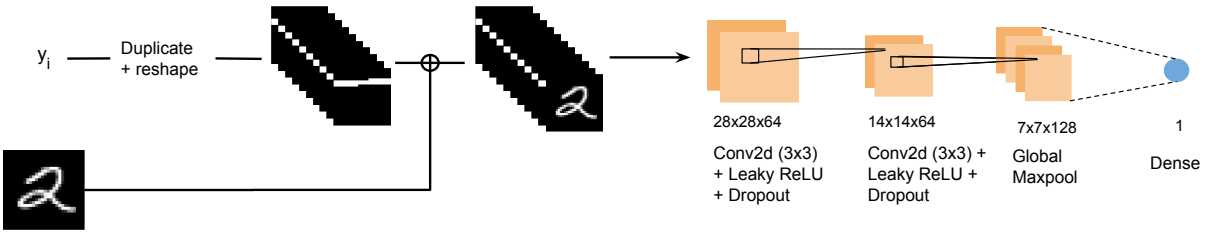


Figure 9: The DCGAN's discriminator architecture using a two-layer CNN. It was conditioned on the digit classes by concatenating the input image with the image representation of its label y_i.

An overview of the conditional DCGAN is shown in Figure 7. The architecture contains a generator G that produces synthetic images from a set of randomly sampled noise vectors with labels $\mathbf{y}$, and a discriminator D that classifies a set of images as real or generated given their labels. The aim of the generator is then to generate images that are classified as real images by the discriminator. We now describe the generator and discriminator model architectures and how they were conditioned on class labels.

The generator model's architecture is shown in Figure 8. It was conditioned on one-hot encoded labels by concatenating them with 128-dimensional random Gaussian noise vectors. The random sampling was done with a seed of 1337 and ensured variability among same-class synthetic images. The noise-embedded labels were processed by a dense layer with 7 x 7 x 138 neurons and reshaped to an image with these dimensions. The final image was then generated using two transposed convolutional layers. The first two layers had 128 channels with 4 x 4 kernels that moved with stride 2. The padding parameter was set such that an image's height and width increased with the factor of the stride. The final layer was a standard convolutional layer and had 1 channel with a 7 x 7 kernel that slid over the input with stride 2. This resulted in a 28 x 28 x 1 synthetic image. The LeakyReLU activation function was applied after each layer in the network, except for the final transposed convolutional layer. Here, the Sigmoid activation was used.

The discriminator model's architecture is shown in Figure 9. The left part of the figure displays how the discriminator was conditioned on the class labels. The one-hot labels were duplicated 10

times and reshaped to 28 x 28 x 10 images. This was concatenated with the input image, resulting in 28 x 28 x 11 content-embedded images. A CNN with two convolutional layers processed this into feature maps, which were reduced to a 128-dimensional feature vector with global max pooling. A single output neuron then returned the probability of the input image being real or not. The two convolutional layers had 64 and 128 channels, respectively, with 3 x 3 kernels that moved with stride 2. The Leaky ReLU activation function was applied after each convolutional layer, as well as dropout with a probability of 0.2 to prevent overfitting.

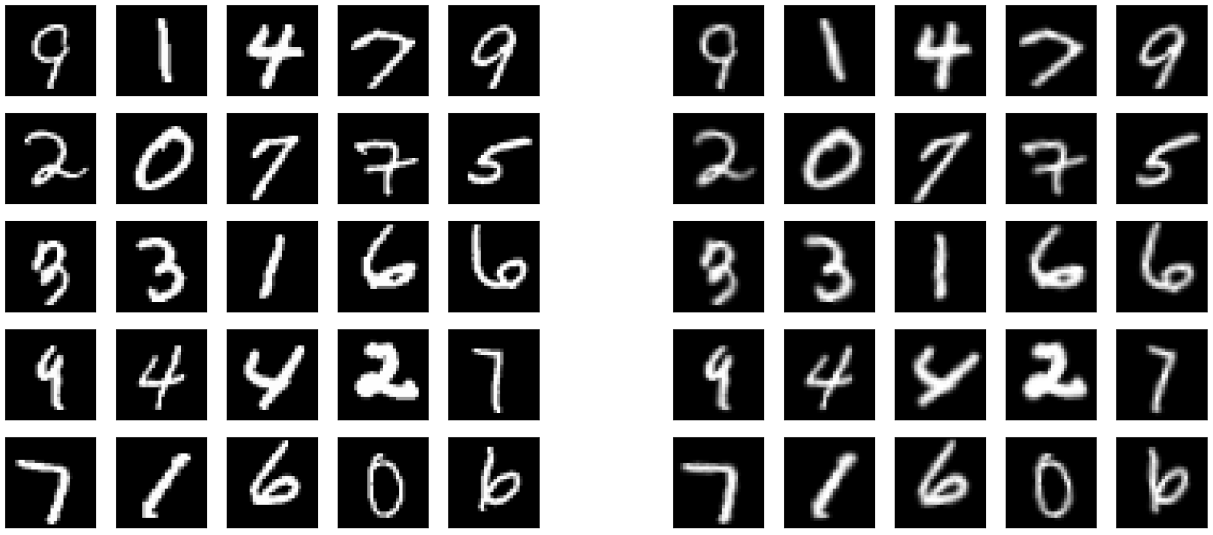
3.3.2 Implementation Details

The DCGAN was trained with the adversarial loss as shown in Equation 4 [58], where $p_z(\mathbf{z})$ is the input noise prior, $\mathbf{y}$ are the ground truth labels, and $\mathbf{x}$ the input images. The generator and discriminator were simultaneously trained such that first, the discriminator is updated by minimizing the first term of $L(D,G)$, and second, the generator is updated by minimizing the second term. The models were trained using the Adam optimizer with a learning rate of 0.0003. The batch size was set to 64.

$$\min_G \max_D L(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log(1-D(G(\mathbf{z}|\mathbf{y})))] \tag{4}$$

3.3.3 Experimental Setup

To maximize the quality of synthetic images, data augmentation was applied. Based on preliminary experiments, each training image was augmented once with a random rotation and shearing, in both x and y axes, within the range of -10 to 10 degrees. A comparison between non-augmented and augmented images can be seen in Figure 10. Considering data augmentation could also reduce synthetic image quality, the DCGAN was trained in two settings: a non-augmented setting with only non-augmented images and an augmented setting with both non-augmented and augmented images.



(a) Non-augmented (b) Augmented

Figure 10: Non-augmented (a) and augmented (b) samples of MNIST.

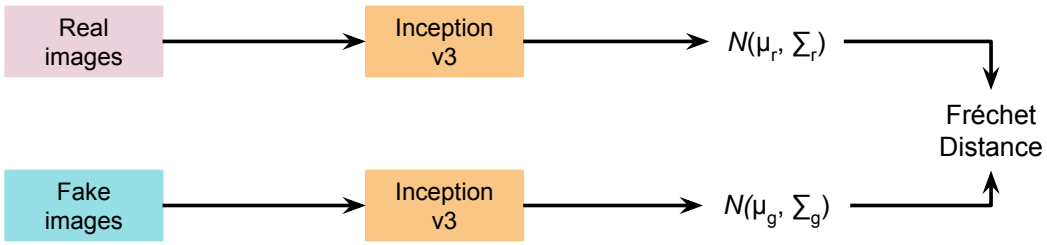


Figure 11: Schematic of the FID calculation

3.3.4 Evaluation

To train the DCGAN, the MNIST-GEN dataset was randomly split such that 90% of the images were reserved for training (27,000 samples), and 10% for validation (3,000 samples). The number of training epochs for the models was based on the training and validation losses, as well as the synthetic image quality. The synthetic image quality was evaluated visually by generating 10 images per digit class every 5 epochs. Additionally, the Fréchet inception distance (FID) between was for quantitative evaluation on the MNIST-GEN validation set. Because calculating the FID is computationally expensive, this was only computed from when the losses started to converge and when the synthetic images no longer appeared to improve visually.

The FID is the Fréchet distance between the normal distributions with means and covariance matrices μ_r, Σ_r for real images, and μ_g, Σ_g for generated images. These are extracted from the feature maps of the Inception-V3 network pretrained on ImageNet [59]. A schematic overview is shown in Figure 11, and the detailed computation is given in 5. The FID is not an optimal metric for image quality, however. It assumes the extracted feature maps are normally distributed and suffers from biases due to the feature maps' high dimensionality. Additionally, considering handwriting specifically, the extracted features may be sub-optimal as the inception-V3 network is trained on natural images. Nevertheless, it is standard to evaluate the synthetic image quality in HTG with the FID and a study on various evaluation metrics for HCG is outside the scope of this thesis.

$$d(\mathcal{N}(\mu_r,\Sigma_r),\mathcal{N}(\mu_g,\Sigma_g))^2 = ||\mu_g-\mu_r||_2^2 + Tr(\Sigma_g+\Sigma_r-2(\Sigma_g\Sigma_r)^{\frac{1}{2}}) \tag{5}$$

3.4 Self-Supervised Character Recognition

3.4.1 Model Integration

After having established the HCR network architecture in Section 3.2 and the generative model in Section 3.3, we can now consider integration into the image-based self-supervised HCR framework. We emphasize the generator model was pretrained and will not be further trained in the self-supervised setting. This prevents potential training instabilities and allows us to focus on the framework's implementation and loss functions. To then integrate the generator and HCR models, the compatibility of their inputs and outputs has to be ensured on two fronts: image preprocessing and label encoding. The next section will address another important component: the loss function. A schematic overview of the image-based self-supervised framework given these components is shown in Figure 12.

Image preprocessing. To compare the input and synthetic images with a loss function, their pixel values need to be on the same scale. To ensure this, we applied identical processing to the

HCG and self-supervised HCR pipelines. This is also most efficient w.r.t. computational costs of the framework.

Label encoding. The fact that the DCGAN was conditioned on one-hot labels simplifies the integration of the pretrained into the self-supervised framework. It enables us to directly input the HCR-predicted class probabilities into the generator for producing the synthetic images. If the generator was not conditioned on one-hot labels, additional operations would have been necessary for decoding and re-encoding the HCR predictions. This can be complicated since these operations would need to be differentiable. Moreover, information about the classes with lower probability predictions would be lost. Therefore, using one-hot encoded labels increases the computational efficiency and uses all available information.

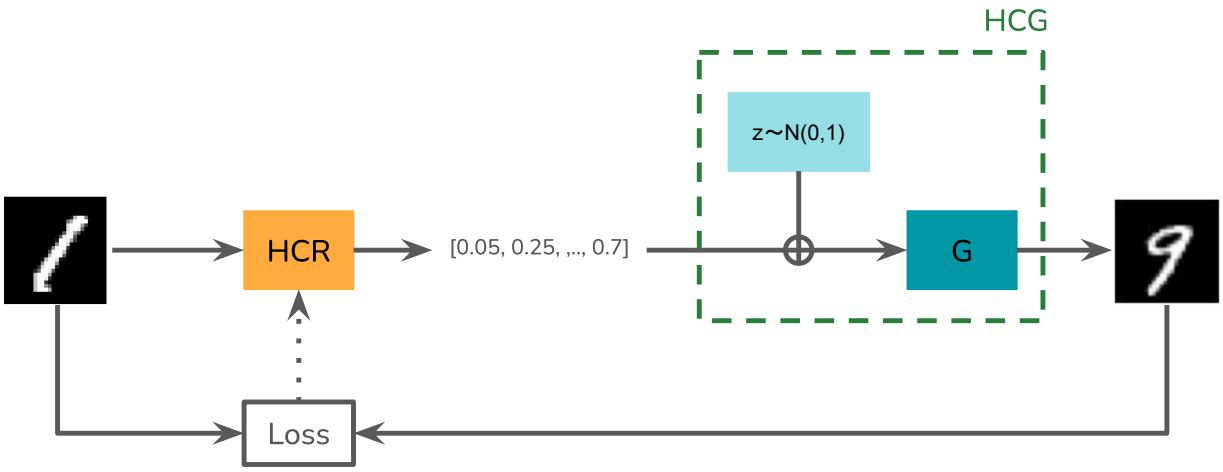


Figure 12: Schematic overview of the image-based self-supervised HCR framework. The generator G is pretrained with the DCGAN architecture as described in Section 3.3.

3.4.2 Loss Functions

An appropriate loss function for self-supervised HCR is one that is minimized when the input and synthetic images contain the same digit. If the loss does not capture the differences in character shapes well, there is a risk of the HCR network exploiting the generator to produce images that, while minimizing the loss, do not contain legible or correct digits. Following the same reasoning, image-based self-supervision, as proposed in this thesis, can be posed as an image reconstruction problem; the HCR network needs to learn to make predictions such that the generator produces images akin to the input images.

Posing self-supervised HCR as an image reconstruction problem, we can take inspiration from loss functions used in this field. Specifically, loss functions were considered that are used to train autoencoders, a type of generative model where an encoder maps images to a latent space, and a decoder reconstructs the images from this. The performances of several loss functions for such models were studied by Khare et al. [60] on various datasets, including the MNIST dataset. They showed that the Mean Squared Error (MSE) and the Binary Cross-Entropy (BCE), in particular, resulted in images with the best quality. Therefore, these objective functions were considered for image-based self-supervised HCR. A third objective function that was considered is the perceptual loss,

which measures differences in style and content between images and showed good results for image generation tasks [61]. These functions are detailed as follows.

MSE The MSE loss penalizes larger differences between pixel values more heavily, being the mean of the squared pixel differences. The MSE is given by Equation 6, where H and W are the image height and width, and $\hat{y}$ and y are the synthetic and input images, respectively.

$$L_{MSE}(\hat{y}, y) = \frac{1}{HW}||y - \hat{y}||_2^2 \tag{6}$$

BCE Like the MSE loss, the BCE loss penalizes larger differences between the image pixels more heavily, but does so by a logarithmic penalty. The BCE loss for an image is given by Equation 7.

$$L_{BCE}(\hat{y}, y) = -\frac{1}{HW}\sum_{i=1}^{W}\sum_{i=1}^{H}[y_{i,j}log(\hat{y}_{i,j}) + (1 - y_{i,j}) \cdot log(1 - \hat{y}_{i,j})] \tag{7}$$

Perceptual loss [61] Unlike the MSE and BCE losses that directly compare pixels of the input images, the perceptual loss operates on a higher level: it compares pixels of extracted feature maps. In this manner, the perceptual loss encourages the generated images to be perceptually similar to the input images. The extracted feature maps are the activations of the jth layer of the VGG-16 network, pretrained on ImageNet. The perceptual loss is computed as in Equation 8, where C is the number of channels of the feature maps and ϕ the pretrained VGG-16 network.

$$L_p(\hat{y}, y) = \frac{1}{C_j H_j W_j}||\Phi_j(\hat{y}) - \Phi_j(y)||_2^2 \tag{8}$$

To determine which jth layer to use for feature extraction, feature maps were extracted at various layers of the VGG-16 network and used to train k-nearest neighbors (kNN) classifiers (k = 5). Specifically, the convolutional layers before each max pooling layer were considered, except for the first max pooling layer. These were chosen based on an initial exploration of t-SNE dimensionality reduction. Hence, feature maps were extracted from the 4th, 7th, 10th, and 13th convolutional layers, which correspond to the 5th, 9th, 13th, and 17th layers in the pretrained VGG-16 network provided by TensorFlow [53].

The kNN classifiers were trained with 5-fold cross-validation on the MNIST-HCR dataset and evaluated with accuracy. The results in Table 2 show that the extracted feature maps at the 7th convolution resulted in the highest accuracy ($98.97\% \pm 0.11\%$), while those extracted at the 13th convolution resulted in the lowest accuracy ($93.90\% \pm 0191\%$). These results reflect the class separability of the feature maps visualized with t-SNE, shown in Figures 13, 14, 15, and 16, respectively. Specifically, much overlap between classes can be observed at the 13th convolution's feature maps (Figure 16), and the least at the 7th convolution's feature maps (Figure 14). Based on these results, the activations of feature maps extracted at the 7th convolutional layer were used to compute the perceptual loss.

Table 2: Results of kNN classifiers trained with 5-fold cross-validation on the MNIST-HCR dataset. The models were trained on the activations of the feature maps extracted at the *n*-th convolution with the VGG-16 network pretrained on ImageNet

*N*th convolution	Validation accuracy (%)
4th	97.80 ± 0.17
7th	98.97 ± 0.11
10th	98.42 ± 0.10
13th	93.90 ± 0.19

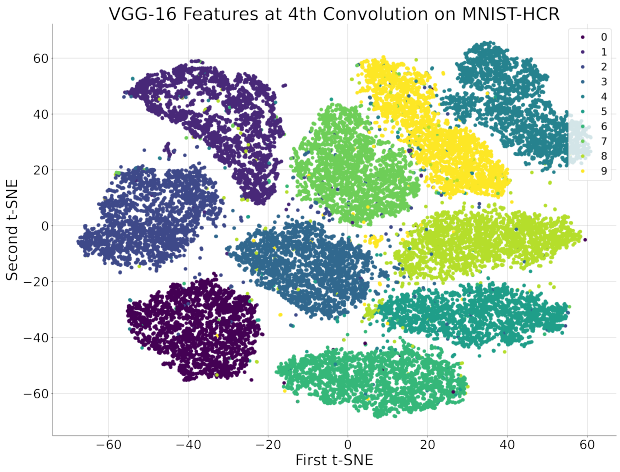


Figure 13: T-SNE of VGG- 16 features maps at the 4th convolutional layer.

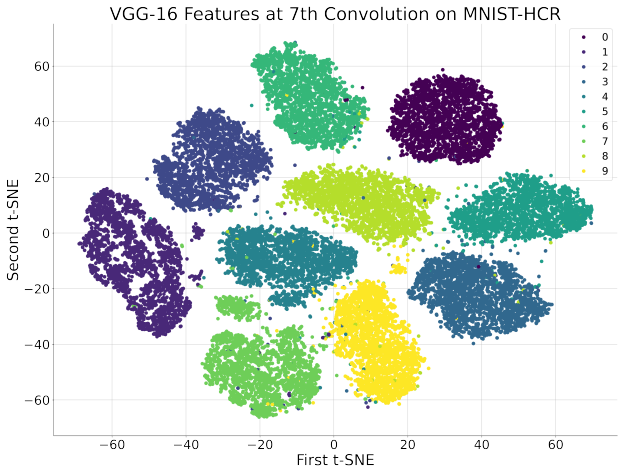


Figure 14: T-SNE of VGG- 16 features maps at the 7th convolutional layer.

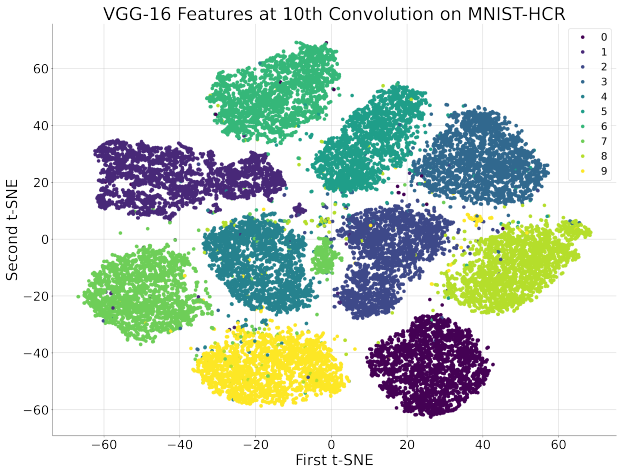


Figure 15: T-SNE of VGG- 16 features maps at the 10th convolutional layer.

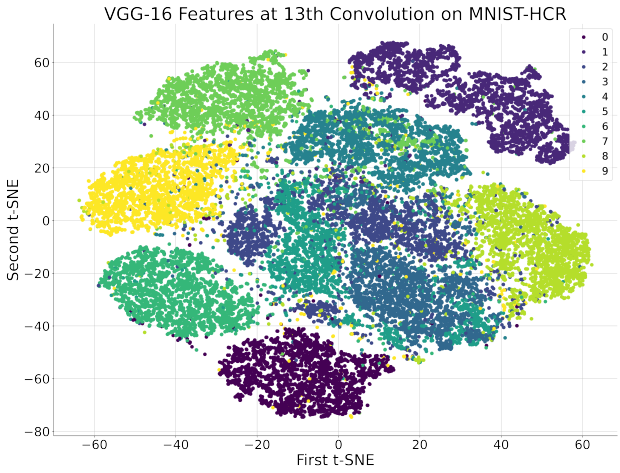


Figure 16: T-SNE of VGG- 16 features maps at the 13th convolutional layer.

3.4.3 Implementation Details & Evaluation

The image-based self-supervised models were trained with the MSE, BCE, and perceptual loss functions and used the Adam optimizer. The initial learning rate was set to 0.001 for all models. Based on preliminary experiments, a multi-step learning rate scheduler was used that decreased the learning rate to 0.0003, 0.0001, and 0.00001 after 1000, 3000, and 5000 iterations, respectively. The batch size was set to 256.

The self-supervised models were trained on the MNIST-HCR dataset with 5-fold cross-validation. Early stopping was applied to the validation loss with a patience of 5 epochs to prevent overfitting. Following the supervised setting, the models were evaluated with accuracy, and on test data the confusion matrix, too. Moreover, the models' test performances were obtained by taking the mean of the predicted class probabilities from the models resulting from each k-fold.

3.5 Results

3.5.1 Supervised Handwritten Character Recognition

The supervised HCR model was trained with 5-fold cross-validation on MNIST-HCR and tested on an unseen test dataset. The test results were obtained by taking the mean of the Softmax probabilities from the models resulting from each k-fold. The evaluation was done with accuracy, and on the test set, the confusion matrix was used as well. To contextualize the model's performance, its test results were compared to DropConnect [55], a state-of-the-art model.

Table 3 shows the validation and test accuracies of our supervised HCR model and DropConnect. Considering the validation accuracy for our HCR model, there was only a small variation between k-folds, which can also be observed in the loss curve shown in Figure 17. For the performance on the test set, the confusion matrix in Figure 18 shows that the most misclassified samples were images of the digits 7 and 8. Samples of the digit 0 were least frequently misclassified. Moreover, no clear bias to a particular class can be observed. Comparing the supervised HCR model to DropConnect, the results show that DropConnect had an increased test accuracy (99.79%) compared to ours (98.82%), indicating that the HCR architecture used in this thesis is sub-optimal. We have to consider, however, that next to a different architecture, DropConnect was trained on the full MNIST training data, which was also augmented. Moreover, a learning rate scheduler was used to further optimize model performance.

Table 3: Validation and test results of supervised HCR. The validation results are on the MNIST-HCR dataset.

Model	Validation Accuracy (%)	Test Accuracy (%)
Ours	97.97 ± 0.10	98.82
DropConnect [55]	-	99.79

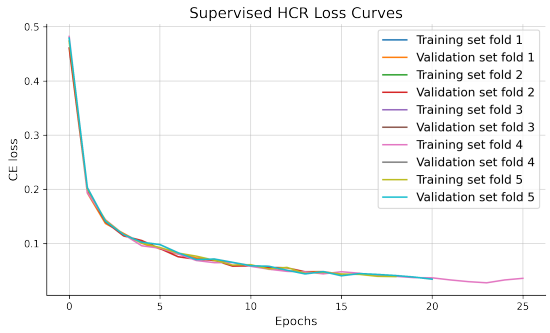


Figure 17: Loss curves of the supervised HCR model for 5-fold cross-validation.

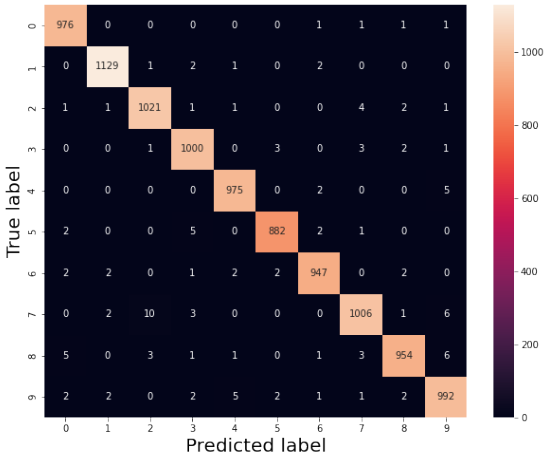


Figure 18: Confusion matrix of supervised HCR on test data.

3.5.2 Handwritten Character Generation

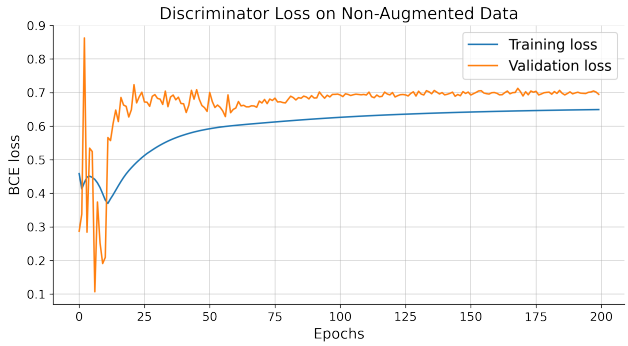
The DCGAN was trained on 90% of the MNIST-GEN dataset (27,000 samples), and validated on 10% (3 000 samples). The models were trained with non-augmented data only in a non-augmented setting, and both non-augmented and augmented data in the augmented setting. The quality of synthetic images was superficially analyzed by generating 10 images per class. Additionally, the models were evaluated quantitatively on the validation set with the FID. To determine the models' training duration, we considered that the losses for the model in the non-augmented setting converged after 200 epochs (see Figure 19). Likewise, as computing the FID is computationally expensive, it was computed in steps of 25 epochs, starting from the 150th epoch. At this epoch, the loss curves for the model in the non-augmented setting started to converge and no clear visual improvements of the synthetic images could be observed.

The loss curves for the generator and discriminator models of the DCGAN in non-augmented and augmented settings are shown in Figures 19 and 20. While the loss curves are not informative with regard to synthetic image quality, they do inform us about the models' stability. The loss curves for the models in both settings show an increasing loss for the discriminators and a decreasing loss for the generators. Additionally, Comparing the non-augmented and augmented settings, it can be seen that the DCGAN in the augmented setting converged much sooner and was overall more stable. However, instability can be observed from epoch 150 in the validation losses, especially in the discriminator's validation loss.

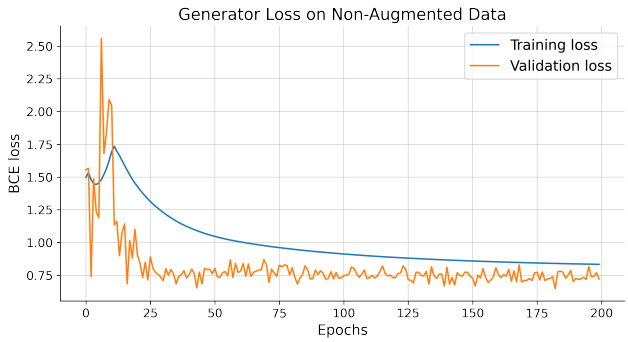
A set of synthetic images is shown across epochs 150, 175, and 200 for models in the non-augmented setting in Figure 21 and in the augmented setting in Figure 22. Common artifacts can be observed between experimental settings and epochs. For one, parts of digits sometimes appear distorted or missing. This is most common for images of the digit 2 in both the non-augmented setting and augmented settings. Another type of artifact is the presence of an extra ink blob, for example in some images of the digit 6 in Figure 21(a), and of the digit 9 in Figure 22(c). These images, however, do not allow for an objective or complete comparison between generative models. Hence, the FID was used for model evaluation.

Table 4 shows the FID for DCGAN models in the non-augmented and augmented settings, where

lower FID indicates improved synthetic image quality. The results then indicate an increased synthetic image quality at epoch 200 compared to epochs 175 and 150 for both settings. Additionally, the DCGANs in the non-augmented setting showed an overall improved synthetic image quality compared to the augmented setting. Lastly, the FID was the lowest for synthetic images generated with the DCGAN at epoch 200 in the non-augmented setting. Hence, we conclude this DCGAN produced synthetic images with the highest quality.

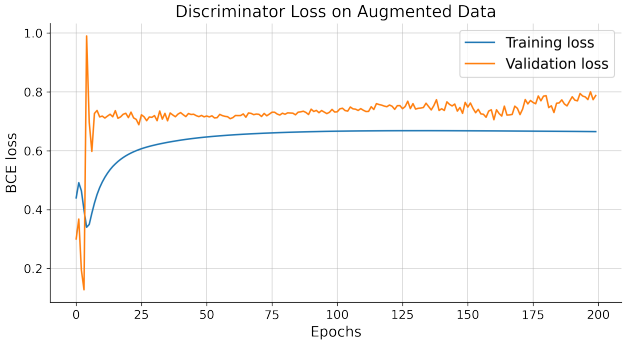


(a) Discriminator loss

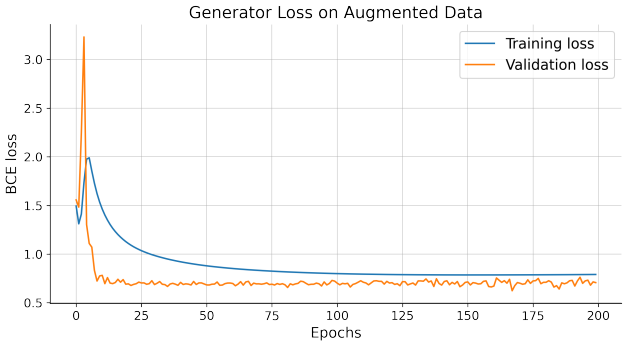


(b) Generator loss

Figure 19: DCGAN loss curves on the non-augmented MNIST-GEN dataset.



(a) Discriminator loss

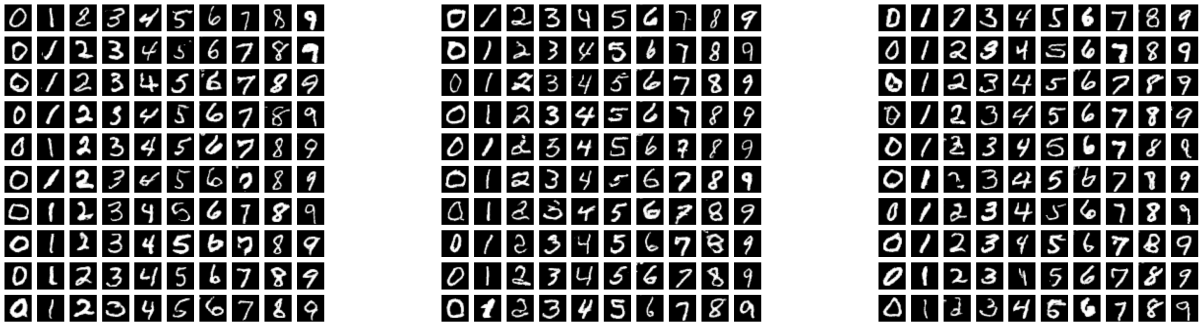


(b) Generator loss

Figure 20: DCGAN loss curves on the augmented MNIST-GEN dataset.

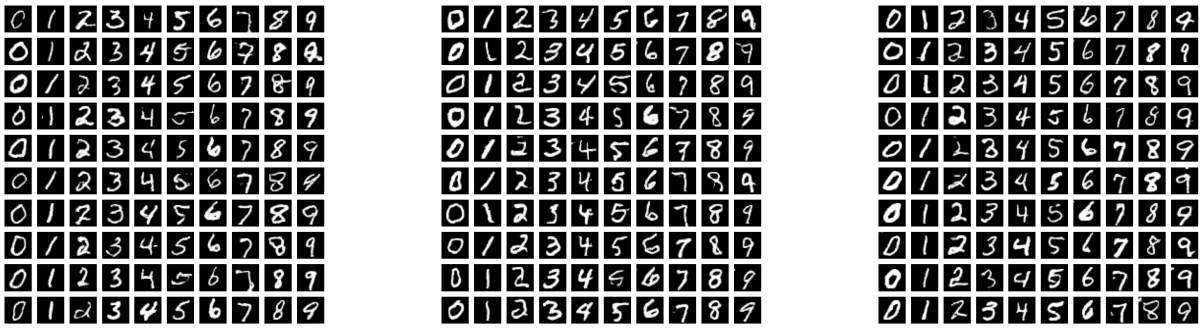
Table 4: FID DCGANs computed between real and synthetic images on MNIST-GEN validation set.

Epoch	FID Non-augmented	FID Augmented
150	12.286	12.317
175	12.034	12.288
200	**11.936**	12.137



(a) Epoch 150 (b) Epoch175 (c) Epoch 200

Figure 21: Synthetic images of the DCGAN trained on non-augmented MNIST-GEN images after 150 (a), 175 (b), and 200 epochs (c)



(a) Epoch 150 (b) Epoch 175 (c) Epoch 200

Figure 22: Synthetic images of the DCGAN trained on augmented MNIST-GEN images after 150 (a), 175 (b), and 200 epochs (c)

Table 5: Validation and test accuracy across loss functions for self-supervised HCR models trained on MNIST-HCR with 5-fold cross-validation.

Approach	Loss function	Validation accuracy (%)	Test accuracy (%)
Self-supervised	MSE	63.11 ± 7.14	66.87
	BCE	60.61 ± 1.95	62.44
	Perceptual	$\mathbf{89.61 \pm 1.29}$	**91.99**
Supervised	CE	97.97 ± 0.10	98.82

3.5.3 Self-supervised Handwritten Character Recognition

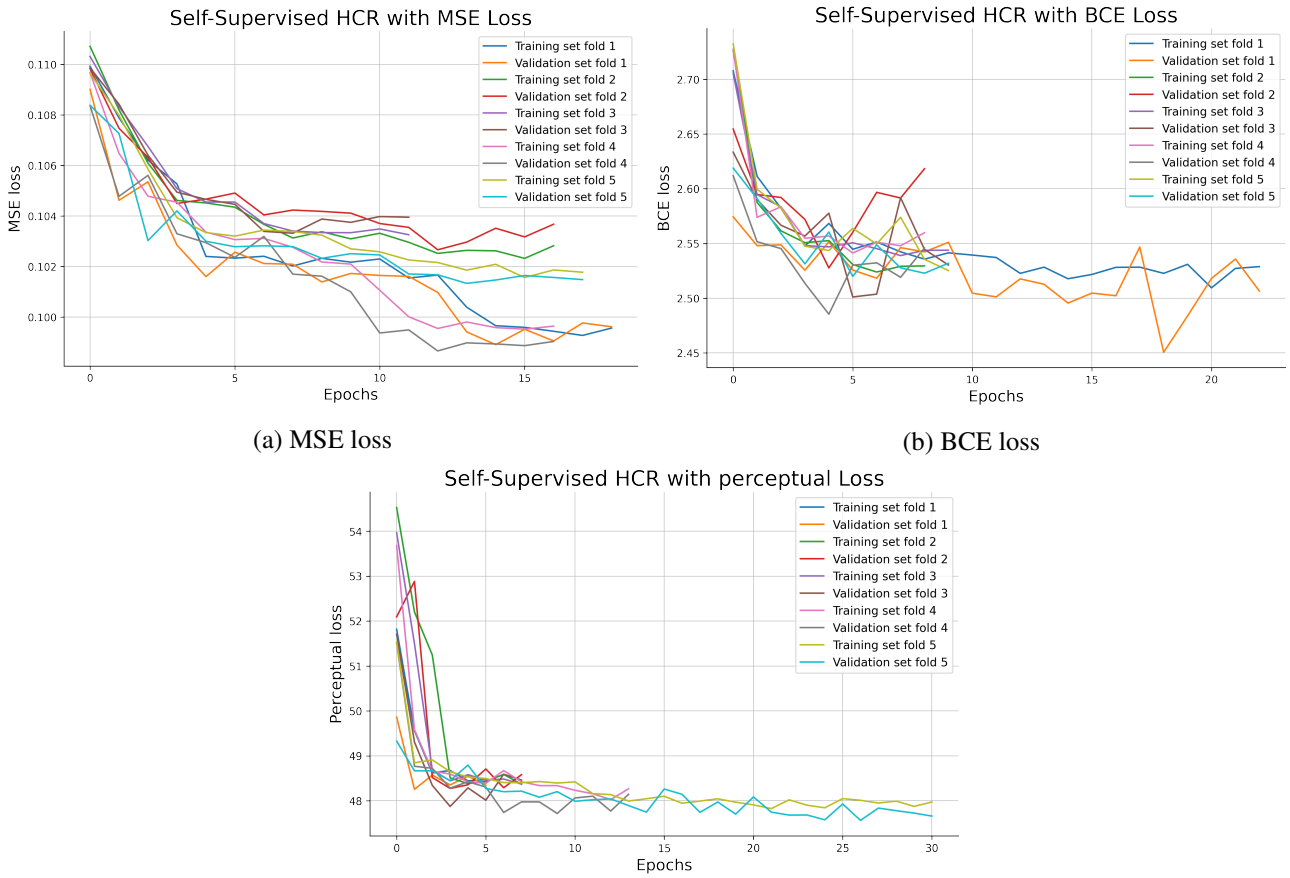
Self-supervised HCR models were trained on the MNIST-HCR dataset with 5-fold cross-validation and tested on a separate, unseen dataset. Results on HCG showed the best-performing DCGAN was trained for 200 epochs in the non-augmented setting. Therefore, this model's generator was used for image-based self-supervised HCR. The self-supervised models were trained with the BCE, MSE, and perceptual loss functions. They were evaluated with accuracy and on the test set with the confusion matrix as well. Test results were obtained by taking the mean of predicted class probabilities between folds. Additionally, the learning processes were observed with the loss curves.

Figures 23a, 23b, and 23c show the loss curves for the self-supervised models trained with the MSE, BCE, and perceptual loss, respectively. The MSE loss curves appear more sensitive to the weight initialization, indicated by the diverging trajectories between the folds. Moreover, they do not show a clear convergence to a certain value. The BCE loss curves also do not converge to a distinct value and show instability within the folds, with sudden increases or decreases. In contrast, the Perceptual loss curves are stable and relatively consistent between and across folds, and appear to converge around the same value.

The observations from the loss curves w.r.t. variability between folds are reflected in the validation accuracies displayed in Table 5. It shows that the MSE loss had the highest variability, as indicated by the standard deviation (7.14%), and the perceptual loss was the lowest (1.29%). Additionally, while the MSE loss resulted in an increased mean classification accuracy relative to the BCE loss, comparing their standard deviations, it was less robust to different initial weights or training and validation splits. On both validation and test data, the self-supervised HCR model trained with the perceptual loss outperformed the other two models. However, compared to supervised HCR (97.97 $\pm$ 0.10% and 98.82%), the MSE and BCE losses led to extremely limited performance, and the perceptual loss achieved only a decent performance with validation and test accuracies of 89.61 $\pm$ 1.29% and 91.99%.

To gain more insight into the models' performances, we consider the confusion matrices in Figures 24a, 24b, and 24c of the MSE, BCE, and perceptual losses, respectively. Here, it can be seen that the model trained with the MSE loss had a strong bias towards digit 1, and misclassified a substantial number of images with the digit 0 as the digit 5 or 6. Additionally, images of the digit 4 were often predicted as a 9. The confusion matrix for the self-supervised model trained with the BCE loss also shows substantial biases, but towards classes 0 and 8. Specifically, the majority of class 1 was incorrectly classified as 8. The perceptual loss, in contrast, did not show any clear biases to a particular class. However, like the MSE and BCE losses, there were relatively frequent misclassifications of images with the digit 5 as the digit 3. Considering the t-SNE data visualization of the MNIST dataset in Figure 5, there was a slight overlap visible between these two classes. This was not the case, however, for the extracted feature maps for the perceptual loss (see Figure 14).

Lastly, the synthetic images with the class predictions are considered for a small set of 16 images of the test set. In this manner, it can be seen whether the predicted classes are indeed generated in the images. These images are shown in Figures 25, 26, and 27 for the self-supervised HCR models trained with the MSE, BCE, and perceptual loss, respectively. For the models trained with the MSE and BCE loss functions, a noteworthy observation is that the synthetic images, in several cases, do not actually contain the digit of the predicted class. This also occurs once in the set of images for the perceptual loss. To investigate this further, we consider the predicted class probabilities for a case from the MSE loss. Specifically, we consider the case in the third and fourth column of Figure 25 where the digit 0 was misclassified as the digit 5, but the image showed a 0 still. The predicted class probabilities are as follows:



(a) MSE loss

(b) BCE loss

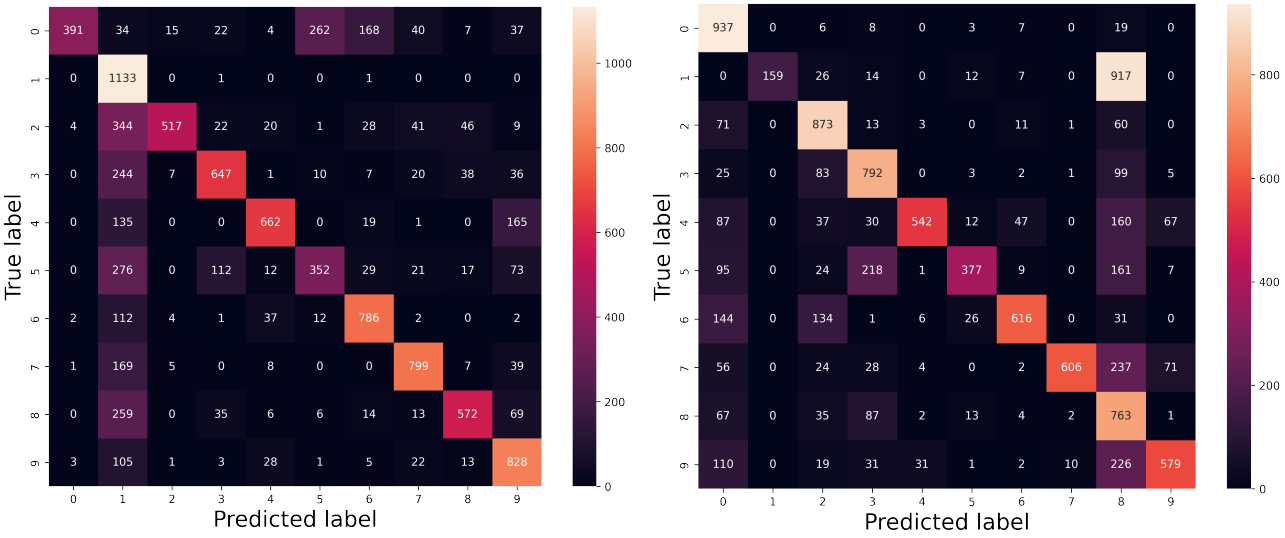
(c) Perceptual loss

Figure 23: Loss curves for the self-supervised HCR models trained with the MSE loss (a), BCE loss (b), and perceptual loss (c) using 5-fold cross-validation.

$$[0.392, 0.0, 0.0, 0.0, 0.0, 0.404, 0.03, 0.005, 0.169]$$

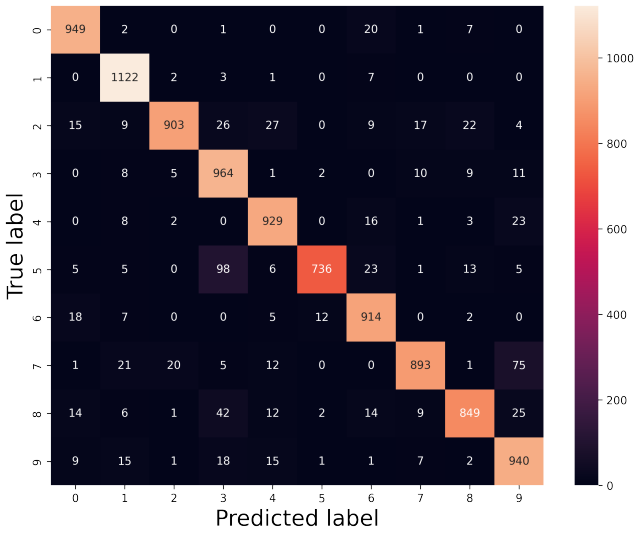
Here, the index of the probability represents the digit class. It can be seen that the model was not confident in its prediction of the digit 5, as the predicted probability for the digit 0 was similar (a 0.012 difference). This ambiguity propagated to the generator, which produced an image that appeared like the digit 0. To confirm this is not a coincidence, we consider another case, but from the BCE loss. In the fourth row and last two columns in Figure 26, the model misclassified the digit 5 as a 3, but the synthetic image ambiguously shows a 0 or 3. This, too, is reflected in the predicted class probabilities, which are as follows:

$$[0.399, 0.0, 0.088, 0.5, 0.0, 0.005, 0.005, 0.0, 0.002, 0.0]$$



(a) MSE loss

(b) BCE loss



(c) Perceptual loss

Figure 24: Confusion matrices on test data for the self-supervised HCR models trained with the MSE loss (a), BCE loss (b), and perceptual loss (c). Test predictions were made by ensembling the HCR models resulting from 5-fold cross-validation.

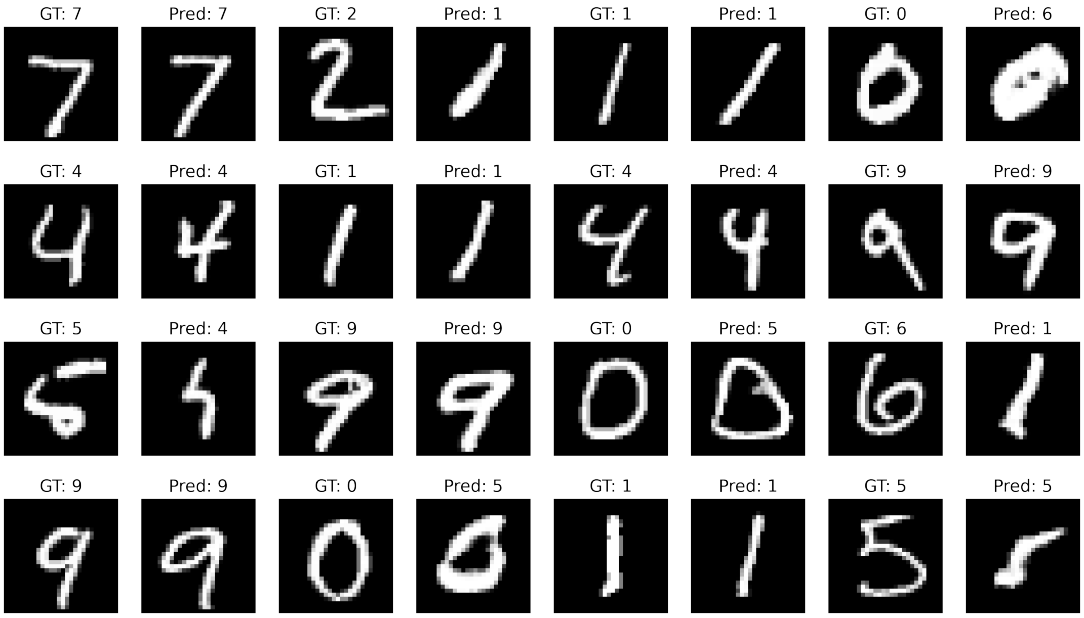


Figure 25: Input images and their corresponding synthetic images that contain the digits predicted by the self-supervised HCR model trained with the MSE loss. The real images are shown in a column first, and then the synthetic images. GT = ground truth.

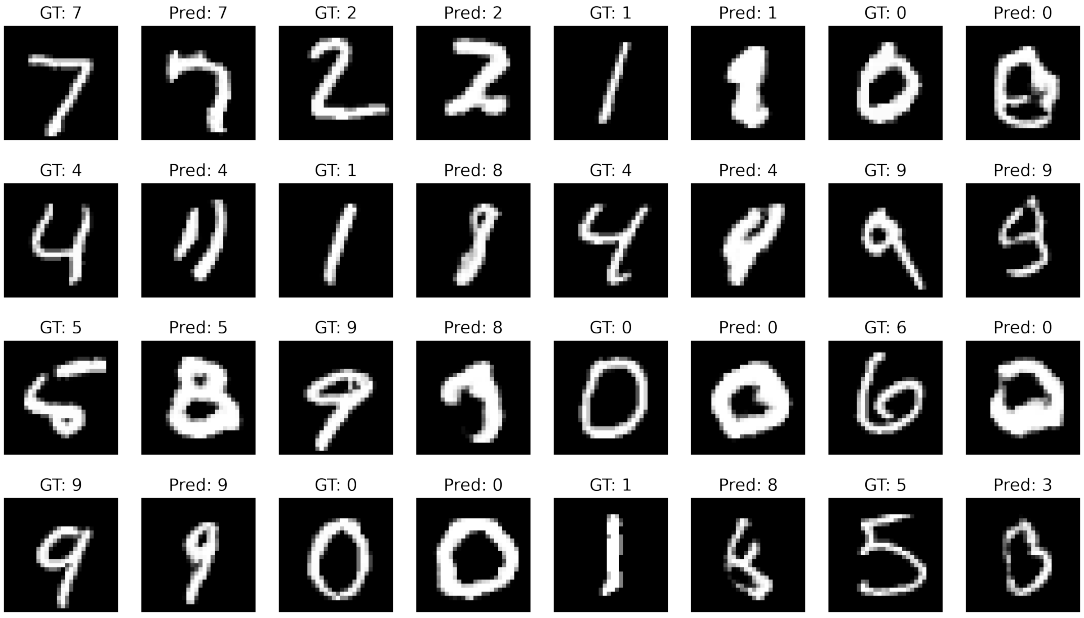


Figure 26: Input images and their corresponding synthetic images that contain the digits predicted by the self-supervised HCR model trained with the BCE loss. The real images are shown in a column first, and then the synthetic images. GT = ground truth.

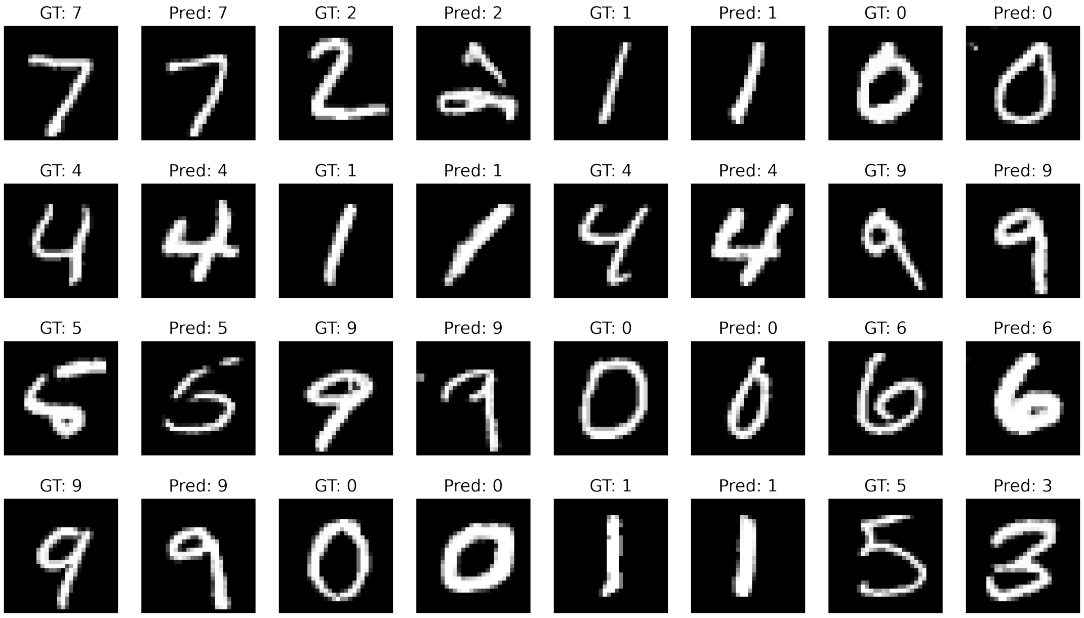


Figure 27: Input images and their corresponding synthetic images that contain the digits predicted by the self-supervised HCR model trained with the perceptual loss. The real images are shown in a column first, and then the synthetic images. GT = ground truth.

3.6 Discussion

3.6.1 Summary & Conclusions

This Chapter served as an initial exploration of image-based self-supervised HTR by applying the framework to the simpler HCR task. Specifically, we used the MNIST dataset to investigate the feasibility of the framework, how it can be implemented, and what type of image-based loss functions could be used to train it. To do so, a DCGAN was trained on half the MNIST dataset and integrated with a simple CNN architecture into the self-supervised framework. The models were integrated by directly inputting the HCR-predicted class probabilities to the DCGAN's pretrained generator. This was possible as the generator was conditioned on one-hot encoded labels. Moreover, it was ensured that the image preprocessing was identical between the HCG and self-supervised HCR pipelines.

Inspired by the task of image reconstruction, we experimented with two low-level (MSE and BCE loss) image-based losses and one high-level (perceptual loss) image-based loss to train the self-supervised HCR system. The self-supervised models were then trained and evaluated on data independent from the DCGAN, using 5-fold cross-validation and an unseen test set. The models' loss curves indicated relevant information to the HCR task was learning. However, this was only limited, especially so for the MSE and BCE losses, which achieved low test accuracies (66.87% and 62.44%, respectively). The perceptual loss led to a much better performance, with a 91.99% test accuracy. This suggests that high-level information is necessary for the image-based self-supervised model to learn substantial information. When compared to the HCR architecture's performance in a supervised setting (98.82% test accuracy), however, the perceptual loss showed a decreased performance. Still, with an accuracy close to 92%, the proposed framework shows potential for further application to HTR.

Given these results, the following conclusions can be made w.r.t. the framework's feasibility, implementation, and loss functions:

- By directly using the predicted class probabilities of the HCR model and using a pretrained generative model conditioned on one-hot encoded labels, substantial information can be learned with image-based self-supervised HCR.
- The proposed image-based self-supervised HCR framework can achieve decent performance, but only when the loss function compares higher-level information extracted from input and synthetic images.

3.6.2 Discussion

The presented methodology has several points of improvement. For one, the results of the HCR architecture in the supervised setting showed that it was not optimal compared to the state-of-the-art model DropConnect. Hence, the performance of the self-supervised models could still be improved by using a different architecture. Considering the differences in the training procedure between our HCR model and DropConnect, data augmentation, a learning rate schedule, and a smaller batch size could also be considered to further improve the self-supervised model's performance. However, the main component of the self-supervised architecture that can be improved upon is the generative model, as the loss computation relies on the quality of the synthetic images.

The generative model was selected considering those trained for 150, 175, and 200 epochs in the augmented and non-augmented settings. This was determined based on the fact that the training loss of the generator in the non-augmented setting started to converge after 150 epochs and that the synthetic images did not appear to improve visually. The augmented setting was not considered separately, and the discriminator and generator's validation losses in this setting showed increased instability in the final 50 epochs. It therefore could have been the case that the DCGANs trained for fewer epochs in the augmented condition (but also in the non-augmented condition) had a lower FID, meaning the selected generator might have been sub-optimal.

Another reason that the performance of the self-supervised HCR models was limited could be that there was no consideration for the writer style in the generation process for synthetic images. For example, if a correct prediction was made, then the synthetic images with the correct digit could have been slanted while the input image it was compared to was not. This may be the reason why the low-level MSE and BCE losses did not perform well in contrast to the perceptual loss. While the perceptual loss does extract higher-level features, it still directly compares the pixels of extracted feature maps, which are also affected by the writing style. In the same manner, the artifacts, specifically more severe character distortions, likely limited the performance as well. For this reason, augmenting the training data with a set of synthetic images could improve performance.

The results of the self-supervised models also showed that the generator produces ambiguous synthetic images due to uncertain predictions being directly input to the generator. It could also be argued, however, that this is a positive property of the architecture. The dissimilarity with the input image will be higher with ambiguous synthetic images, compared to when there is a clear character in the synthetic image. Therefore, it could aid in more confident correct predictions. This also highlights the need for a loss function that can capture the differences in character shapes well. As aforementioned, it should not be too specific on the shape, but more general so that it captures the content information instead of the stylistic information. Given the results and the aforementioned explanations, higher-level loss functions should then especially be considered for the more complex word recognition task.

4 Methods

In Chapter 3, we showed the proposed self-supervised framework was able to learn substantial information for HCR by 1) using a loss function comparing higher-level information through extracted feature maps, and 2) directly inputting the predicted HCR class probabilities to a pretrained generator, which was conditioned on one-hot encoded labels. This Chapter builds upon these findings by applying image-based self-supervised learning to word recognition, posed as a sequence processing task (see Figure 2). Through a series of experiments, we thus address the primary research question of whether the proposed framework can be effectively used for HTR, as well as the secondary questions of how the framework can be implemented, with what loss function it should be trained, and whether it can further improve existing HTR systems. Following our approach to self-supervised HCR, the HTR architecture and HTG model were first established before they were integrated into the proposed framework.

The methodology presented in this Chapter was implemented in Python. Specifically, the network architectures were implemented with PyTorch [62]. The code is publicly available at `https://github.com/Lisa-dk/handwriting-recognition`

4.1 Dataset

For the selection of a dataset, two requirements of the generative model need to be considered w.r.t. the image-based self-supervised framework. First, it needs to be able to produce synthetic images containing the text HTR-predicted texts. Second, the text needs to have the same writer as the input image. This means the HTG model needs to be conditioned on both content and writer style. To enable this, a dataset is needed which contains labels for both the text and writing styles of its images. One benchmark for HTR that has both is the IAM handwriting database [63].

The IAM database consists of 1539 pages of English texts, handwritten by a total of 657 writers. The pages are grayscale PNG images that were scanned with a resolution of 300 DPI and have pixel values ranging from 0 to 255. The writers are labeled as integers, and the text labels are strings in ASCII format. Moreover, the pages were automatically segmented into line and word images. As aforementioned, it was determined to focus on word recognition as the HTR task is more complex on higher levels of abstraction due to the higher number of possible character sequences, additional language structures, and fewer available data samples. Additionally, state-of-the-art HTG models operate on word images. In total, there are 115,320 variably-sized word images in the IAM database. Figure 28 shows six examples of such images.

Following the HTG literature, the training and test partitions of the IAM database provided by Kang et al. [39] were used. We refer to this dataset as IAM. The dataset contains words with a minimum of two and a maximum of seven characters. The punctuation and digits were filtered from

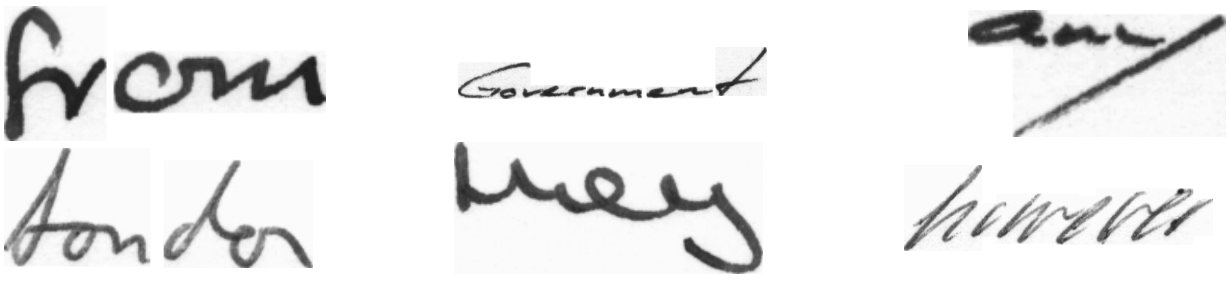


Figure 28: Samples of the IAM database.

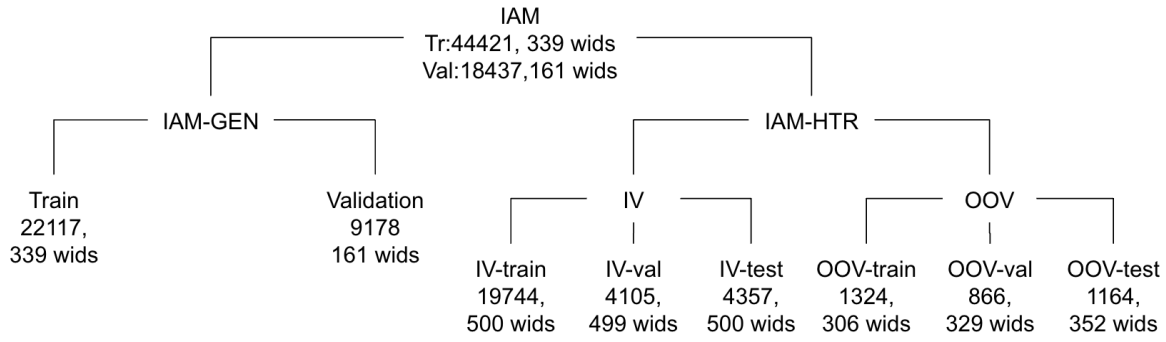


Figure 29: The IAM-GEN and IAM-HTR partitions of the IAM data split by Kang et al. [39]. Both partitions contained 50% of the IAM dataset, maintaining the writer styles' class proportions. The IAM-GEN data splits are writer-independent. The IAM-HTR has in-vocabulary (IV) and out-of-vocabulary (OOV) subsets.

the text labels. The data splits are mutually exclusive w.r.t. writer styles to allow testing for the HTG models' ability to generalize to unseen writer styles. Hence, the training and validation sets contain 339 and 161 writer styles, and 44,421 and 18,437 word images, respectively. Similarly to MNIST, the IAM training and validation sets were each split in half to maintain independence between the HTR and HTG models. This was done by assigning 50% of the samples for each writer to one of two datasets: IAM-GEN and IAM-HTR. IAM-GEN was used for training the generative model, and IAM-HTR for training the HTR models. As the focus of HTR is on correctly predicting the text independent of the writer's style, the writer-independent IAM-HTR training and validation sets were merged and refactored. Consequently, only the IAM-GEN dataset was mutually exclusive w.r.t. writers. An overview of the datasets, including the number of samples and writer identifiers (wids) per subset, is shown in Figure 29.

In the HTR literature, it is repetitively argued that HTR systems struggle to generalize to new, out-of-vocabulary words. However, concrete results supporting this are difficult to find. Therefore, to confirm and further investigate this, the IAM-HTR dataset was split into an in-vocabulary dataset (IAM-HTR-IV), and an out-of-vocabulary dataset (IAM-HTR-OOV), where the latter contained a small set of words separate from the in-vocabulary dataset. Both had their own training, validation, and test sets. The IAM-HTR-OOV dataset included the set of words (3354 images in total) for which there were a minimum of three and a maximum of six samples in the IAM-HTR dataset. In this manner, each data split of IAM-HTR-OOV contained at least one sample of each word. When there were four samples of a word, two were included in the training set, and one in the validation and test sets each. For five samples, two were included in the training and validation sets each, and one in the test set. For six samples, two were included in each IAM-HTR-OOV data split.

The IAM-HTR-IV dataset contained all samples for the set of words that were not in the IAM-HTR-OOV set (28206 samples total). Of these samples, 70% was included in the training set, and 15% in the validation and test sets each. While the samples were selected randomly, it was ensured that each writer's style was present in each of the splits. Only in the validation set one writer style was not included as it had only two samples.

4.1.1 Image Preprocessing

In Chapter 3, it was proposed to use identical image preprocessing between the HCR and HCG pipelines to ensure compatibility between the models. Additionally, it reduces the computational costs of image-based self-supervised HTR by minimizing the number of operations for the loss computation. This is more so relevant for word recognition as IAM's word images are larger than MNIST's 28 x 28 digit images. Therefore, the preprocessing was kept identical between HTR and HTG models as well. Because generative models are commonly sensitive to the image preprocessing procedure, the image preprocessing steps of the HTG model used for self-supervised HTR were followed. For the self-supervised HTR experiments in this thesis, GANwriting [39] was used as a generative network. We elaborate on this decision in Section 4.3. The model takes and generates 64 x 216 grayscale images with a pixel range of [-1, 1]. The preprocessing steps can be summarized as follows:

1. Resizing to the networks' input size of 64 x 216 pixels.
2. Intensity normalization to [0, 1] through division by 255.
3. Inversion.
4. Standardization with the z-score using a mean and standard deviation of 0.5. Consequently, pixel values ranged from -1 to 1.

To resize the word images to the desired height and width of 64 and 216 pixels, the images were resized to the height ratio (i.e. $h_{new} = \frac{h_{img}}{64}$). However, if the width of the resulting images was greater than the desired width of 216 pixels, the superfluous part was removed, regardless if any characters were present. This resulted in 1) a loss of information and 2) incorrect labels. Therefore, the first preprocessing step was adapted. This was done by first padding the height of small images, and then resizing the images by maintaining the aspect ratio. These steps can be detailed as follows:

1. **Height padding** Word images less than half the desired height of 64 pixels were padded with Δ_h white pixels such that the images were centered vertically. This was computed as in Equation 9, where h is the image height. When resizing small word images as described in 2., this prevents a 'zooming in' effect of large characters, and a 'zooming out' effect of small characters. While this did not affect many word images, it alleviated some scaling differences between characters in word images after resizing. Examples of this are shown in Figure 30.

$$\Delta_h = \lfloor \frac{64}{h} \rfloor \cdot (1 - \frac{h}{64}) \tag{9}$$

2. **Resizing to 64 x 216** Images were resized maintaining the aspect ratio, following de Sousa Neto et al. [18]. This was based on the maximum ratio f of an image's height h and width w to the desired values. The new height and width were then computed as in Equation 10, where d is the image dimension's size, d_{new} is its new size, and d_t its target size. This ensured the resized image did not exceed the dimensions of 64 x 216. If either the height or width was smaller than the intended size, padding was applied with the most frequent pixel value in the image.

$$d_{new} = max(min(\frac{d}{f}, d_t), 1) \tag{10}$$

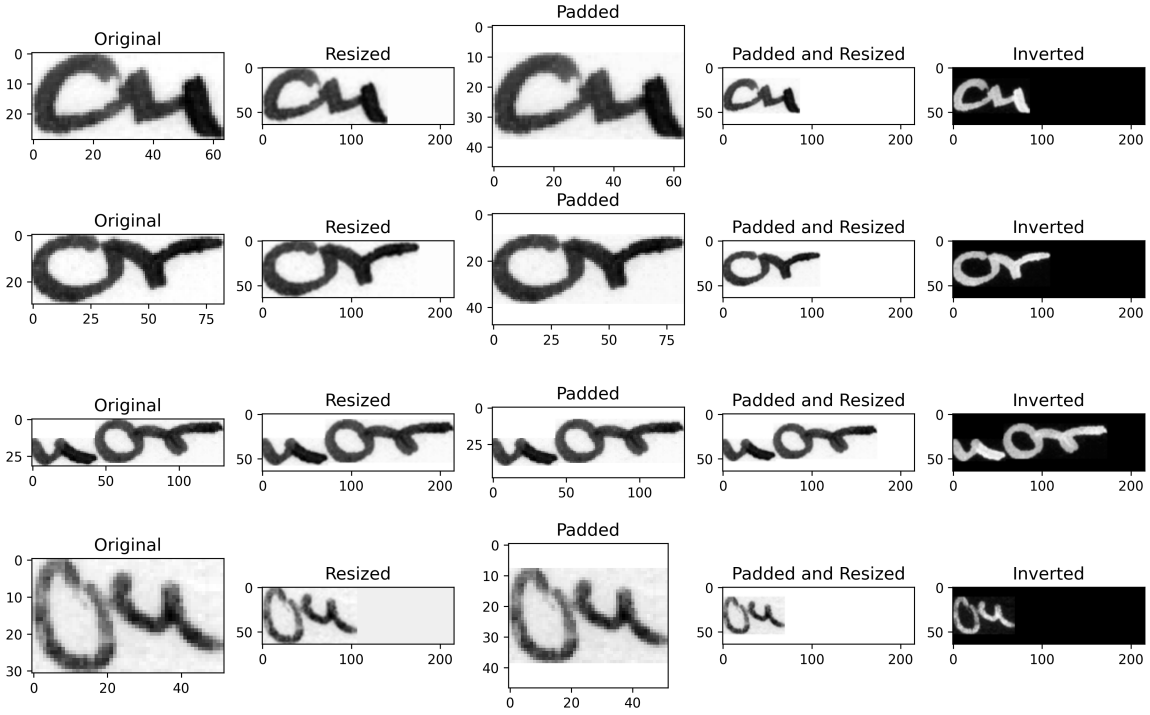


Figure 30: Examples of how extra padding to the images' heights prevent a 'zooming in' effect in resized images.

4.1.2 Text Label Encoding

To determine how to encode the text labels, it had to be considered that in the self-supervised HTR framework, GANwriting's generator has to produce word images with the HTR-predicted class probabilities. As in the image-based self-supervised HCR framework, it was most efficient and simple to directly input the HTR predictions to the generator. Moreover, this allows for the consideration of the HTR model's confidence in its predictions. Consequently, GANwriting's generator had to be conditioned on one-hot encoded text labels. However, these encoded labels should represent the same characters for the pretrained generator and the HTR architecture. This is especially relevant when using a pretrained HTR model. If the indices of the HTR model's predictions represent different characters than the one-hot labels the generator is conditioned on, the synthetic images would contain characters different than those that were predicted. Therefore, the same vocabulary should be used to encode the labels for the HTG and HTR models.

The vocabulary contained the 26 alphabetical characters in both upper and lower case. Additionally, it contained start, end, and padding tokens as well as a token used to represent characters the computer could not recognize. The vocabulary thus consisted of 56 characters in total. The start and ending tokens were required by GANwriting, and the padding token was needed to match the dimension of the label representation with the number of time steps predicted by the HTR model. Unrecognized characters were a rare occurrence, and the token mostly served for future extensions to other datasets. The labels were encoded by firstly adding the start and end tokens, secondly mapping the characters to their corresponding indices in the vocabulary, thirdly adding padding tokens, and lastly mapping the labels to their one-hot vectors. The implementation for this was adapted from [18].

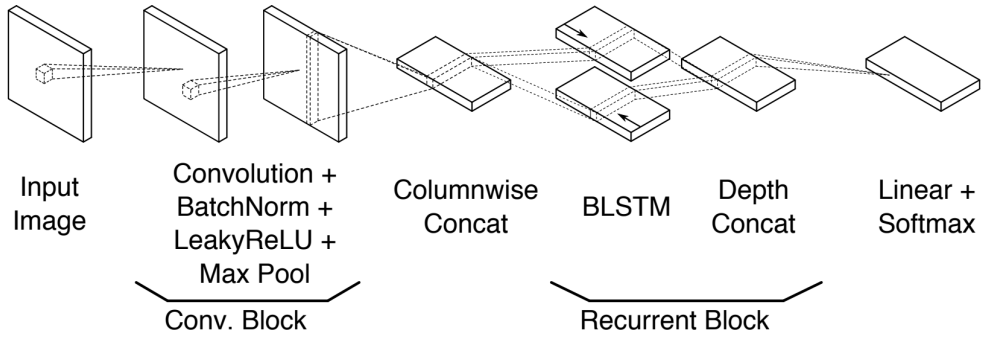


Figure 31: Overview of the Puigcerver model architecture. Figure obtained from [16].

4.2 Supervised Handwritten Text Recognition

4.2.1 Model Architecture

Given that the proposed self-supervised framework involves two deep learning models and an image-based loss, it is computationally expensive. Therefore, it would be ideal for the HTR model to have a simple, lightweight architecture that is also able to achieve performance similar to the state-of-the-art. Initially, we intended to use the HTR-Flor model [18], which meets both of these criteria. It contains only 0.8M parameters, compared to 64M in the state-of-the-art model TrOCR, and performs similarly. However, HTR-Flor was implemented in TensorFlow, which was incompatible with GANwriting's implementation in PyTorch. We found this to be a general discrepancy between HTR and HTG. The pretrained generator was converted to TensorFlow with ONNX, but the synthetic images produced by the converted generator differed from those generated by the non-converted model. Rather than investigating the potential causes, we aimed to focus on successfully implementing the self-supervised framework. As GANwriting was relatively complex, and GANs are prone to non-convergence, mode collapse, and vanishing gradients, it was decided to implement a simpler HTR architecture in PyTorch instead of implementing GANwriting in TensorFlow. Consequently, the relatively simple CNN-BLSTM by Puigcerver [16] was used. To implement this in PyTorch, the code structure of de Sousa Neto et al. [18] was adopted.

The Puigcerver CNN-BLSTM architecture is shown in Figure 31. It contains five convolutional blocks for feature extraction and five recurrent blocks for sequence processing. The feature maps extracted with the convolutional blocks are reduced to one-dimensional feature vectors through column-wise concatenation. The convolutional layers had 16, 32, 48, 64, and 80 channels, respectively. Additionally, max pooling with a 2 x 2 kernel was applied only to the first three convolutions, and dropout with a probability of 0.2 was applied only to the last three convolutions. Regarding the recurrent blocks, dropout was applied after each BLSTM layer with a probability of 0.5. The number of hidden units of the BLSTM layers was fixed to 256.

During initial experimentation, the Puigcerver model combined with the aforementioned image preprocessing and text label encoding performed worse than reported in [16]. Likely, this resulted from applying the model to word recognition instead of line text recognition. To then obtain better performance, the Puigcerver architecture was adapted by replacing column-wise concatenation with column-wise max pooling. Retsinas et al. [64] argue that this improves HTR performance as it introduces translation invariance with regard to the characters' vertical position in the images. The adaptation also reduces the size of the feature vectors from B x W x H x C to B x C, with B, H, W, and C being the batch size, height, width, and number of channels of the feature maps, respectively.

4.2.2 Implementation Details

Puigcerver [16] trained their network with the CTC framework [13], which is commonly used for line-level HTR. Normally, a network outputs a conditional class probability distribution across time steps. The probability of a single label sequence, or alignment, is then the product of its predicted character probabilities. In the CTC framework, an extra blank symbol such as ε is introduced to model consecutive characters. In this framework, repeated characters are first collapsed into a single character, and then blank symbols are removed. Without the blank symbol, a prediction for an input image with, e.g., the word "peers" could be "ppeeeersss", which would be collapsed to "pers". With a blank symbol, consecutive characters can be prevented from being collapsed: "ppεeeεerss" would be merged to "pεeεers". With the removal of the blank symbol, the final prediction would be 'peers'. In this manner, there is no need for an explicitly defined alignment between the input and text labels. The probability of a possible label sequence l for an input x, $p(l|x)$ is then the sum of the probabilities of each possible alignment. Given this, the CTC loss is expressed as in Equation 11.

$$L_{CTC} = -log(p(l|x) \tag{11}$$

As aforementioned, during the initial experimentation of the Puigcerver model with the CTC loss, the results were relatively low compared to those reported in [16]. The CTC loss may be a better fit for higher-level HTR since this has less explicit alignments. Hence, we also experimented with the Cross-Entropy (CE) loss, given by Equation 3. Its multiclass and probabilistic nature make it suited for the sequential word recognition task. Moreover, with the application of a pretrained HTR model for self-supervised word recognition in mind, the CE loss function is more compatible with the pretrained generator as it does require an extra blank symbol in the vocabulary.

Given the adaptation of the architecture and the two loss functions, a total of four experiments were conducted, where models were trained with either column-wise concatenation or max pooling, and either the CTC or CE loss. In all experiments, the network was trained with the Adam optimizer, using an initial learning rate of 0.0001. The learning rate was reduced by a factor of 0.2 when the validation loss did not increase for at least 5 epochs. The batch size was set to 16 samples.

4.2.3 Evaluation

The supervised HTR models were trained on the IAM-HTR-IV training set and evaluated on the HTR-IAM-IV and HTR-IAM-OOV validation and test sets. Early stopping was applied with a patience of 10 epochs on the IAM-HTR-IV validation set. To prevent tuning on the test set, the best model architecture was selected based on the validation results. To obtain the predicted word, the sequences of predicted character probabilities were decoded greedily by taking the argmax at each time step; the predicted character at time step t was the most likely character at this time step. The resulting tokens were first mapped to their corresponding characters in the vocabulary, after which the start, end, padding, and unrecognized characters were removed.

To evaluate the models, the character error rate (CER) and word error rate (WER) were used. These are the standard evaluation metrics used in HTR literature. The CER and WER were computed over the decoded predicted character sequences and ground truth text labels. The CER is the Levenshtein distance between the two strings divided by the number of characters N in the ground truth label, usually expressed as a percentage (see Equation 12). The Levenshtein distance is the minimum number of character insertions I, deletions D, and substitutions S necessary to transform one string into another.

$$CER = \frac{I+D+S}{N} \cdot 100 \tag{12}$$

The WER is computed as the Levenshtein distance on the word level between two strings of text divided by the total number of words in the ground truth. For word recognition, this becomes the percentage of incorrectly predicted samples W_I to the total number of samples W_T in the dataset (see Equation 13).

$$WER = \frac{W_I}{W_T} \cdot 100 \tag{13}$$

4.3 Handwritten Text Generation

4.3.1 Model Architecture

Recent research on HTG shows a growing trend towards diffusion models, which have demonstrated impressive image quality w.r.t. content and particularly, writer style. Diffusion models may replace GANs as the state-of-the-art in HTG, but for usage in combination with external models such as in the proposed framework they significantly increase the computational costs and, consequently, the training times. State-of-the-art GANs, on the other hand, are difficult to integrate with other models due to their high complexity. Balancing image quality and model complexity, we used GANwriting [39] for the generative model. Moreover, its implementation is publicly available[2].

GANwriting leverages four types of models: a generative model H, a discriminator D, a writer classifier W, and a handwritten text recognizer R. Each of these models has its own functionality: the generative model produces synthetic images, and the discriminator, writer classifier, and handwritten text recognizer, respectively, enforce realism, writer style, and legible text in the synthetic images. An overview of all components of GANwriting is shown in Figure 32.

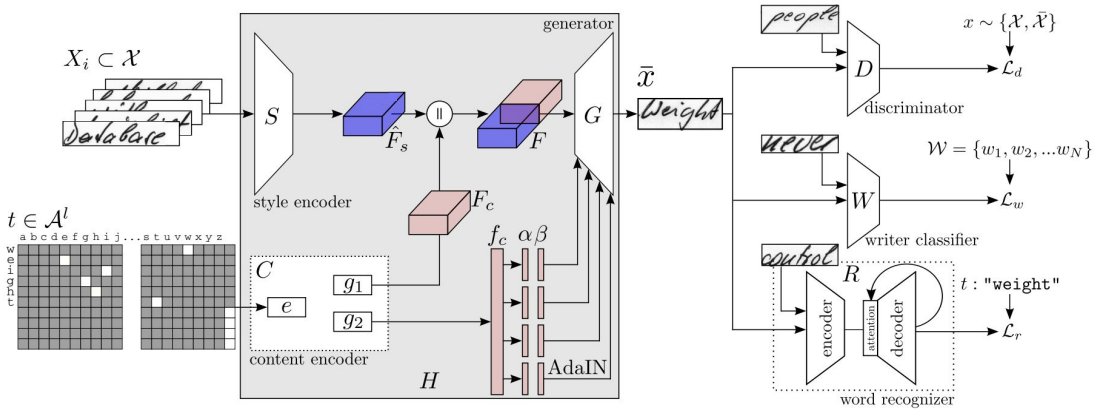


Figure 32: An overview of the GANwriting architecture, adapted from [39]. Here, $\mathcal{X}$ is the set of word images, $\mathcal{W}$ is the set of writer identifiers, and $\mathcal{A}$ is the vocabulary. The generator is conditioned on style and content representations extracted by the style- and content encoders. The discriminator, writer classifier, and word recognizer enforce realism, writer style, and text legibility in the synthetic images.

[2] https://github.com/omni-us/research-GANwriting

The generative architecture generates images using a style encoder S, a content encoder C, and a conditional generator G. The style encoder has a VGG-19-BN architecture that operates on a set of K word images written by the same writer (i.e. style images, X_i) and produces a style representation $\hat{F}_s$. The style encoder encodes a writer's style by learning to map the style images to a latent space, such that, the textual contents and writer's style are disentangled. To mimic within-writer variability, the set of style images was obtained by shuffling the full set of images corresponding to a single writer, and selecting the first K images. If less than K word images were available, the available samples were duplicated.

The content encoder consists of three components: an embedding layer e, and two fully connected neural networks (FCNs) g_1 and g_2. The embedding layer is a linear layer that maps each character c in the text input t to a one-dimensional character representation, which is input to the FCNs. Both FCNs have three dense layers with 1024, 2048, and 4096 neurons, respectively. The ReLU activation function and batch normalization are applied after each layer. FCN g_1 forms a character-wise content representation F_c by processing each embedded character $e(c)$ individually and stacking them together. F_c is then concatenated with the style representation $\hat{F}_s$ to a single feature representation F. This would allow for the generation of images with OOV words. FCN g_2 forms a global content representation f_c by processing the concatenated character embeddings. This is then split into four pairs of parameters to be input to intermediate layers of the generator network. To enable integration of the generative architecture with the HTR network, the embedding layer was removed and the one-hot encoded text labels were directly input to g_1 and g_2.

Instead of directly being input to the generator, the feature representation F is first processed by a dense layer l with 512 neurons. The generator network then consists of two residual blocks (512 channels, 3 x 3 kernels, stride=1) with adaptive instance normalization (AdaIN) layers [65] that align the channel-wise mean and variance of $l(F)$ with the partitions of f_c. Following the residual blocks are three nearest-neighbor up-sampling and convolutional layers (256, 128, 64 channels, 3 x 3 kernels, stride=1) with the ReLU activation function, and a final convolutional layer (1 channel, 7 x 7 kernel, stride=1) with the tanh activation function that outputs the synthetic image $\bar{x}$.

The discriminator predicts the probability of an input image being real or synthetic. In GANwiting, the discriminator network consists of a convolutional layer with 16 channels, followed by six residual blocks with ReLU activations. Average pooling is applied after each block except for the last. Each of these six residual blocks contains two residual sub-blocks, totaling 12 residual blocks. Each second sub-block has an extra convolution (three total) compared to the first. The convolutions in the six blocks have 32, 64, 128, 256, and 512, and 1024 channels of 3 x 3 kernels with a stride of 1, respectively. Finally, a convolution is applied with 1024 channels and a 2 x 2 kernel with stride 7. This reduces the output size for a single image to 1024 x 1.

The writer classifier enforces the stylistic properties in the synthesized images. It uses the same network architecture as the discriminator model, where the classification layer has 500 channels instead of 1024, corresponding to the number of writers in the dataset.

The text recognizer enforces the desired content to be present in the synthesized images. The network is an attention-based seq2seq model [66]. It has an encoder with a VGG-19-BN architecture and two Bidirectional GRU layers. The decoder is a two-layer, one-directional GRU, followed by a dense layer with 500 neurons, corresponding to the number of writers. The at-

tention mechanism aligns the context feature vector of the decoder with the feature vector from the encoder. For more details on the attention mechanism, we refer to [66] and [39].

4.3.2 Implementation Details

GANwriting was trained end-to-end, which the authors claim leads to improved results. In one training epoch, the weights of each model R, W, D, and H were updated in that order. If one model's weights were updated, that of the other models remained frozen. The text recognizer and writer classifier were trained only on real images with the Kullback-Leibler Divergence (KLD) and CE loss. The KLD loss is given by Equation 14, where l is the number of characters in the encoded ground truth label, $|\mathcal{A}|$ the vocabulary size, y the encoded ground truth label, and $\hat{y}$ the predicted character probabilities.

$$L_R = \sum_{i=1}^{l} \sum_{j=1}^{|\mathcal{A}|} y_{i,j} log(\frac{y_{i.j}}{\hat{y}_{i,j}}) \tag{14}$$

The discriminator was trained on real and synthetic images with the adversarial loss as defined in Equation 4. The generator was trained with a summation of the losses of the text recognizer, writer classifier, and discriminator computed on synthetic images and conditioned inputs, given by Equation 15.

$$L_H = L_D + L_W + L_R \tag{15}$$

To train GANwriting, the proposed hyperparameters by the authors were used. Hence, all four models were optimized with the Adam optimizer. The learning rates were 0.0001 for the discriminator and generator and 0.00001 for the text recognizer and writer classifier. The batch size was set to 8. Different batch sizes were experimented with to decrease training times, but these produced suboptimal results. The style representations were extracted from $K = 25$ style images as opposed to the 15 proposed by Kang et al. [39] to compensate for the reduced number of training samples in IAM-GEN compared to IAM.

4.3.3 Evaluation

GANwriting was trained on the IAM-GEN training set and evaluated with the validation set, following the same experimental setup as proposed in [39]. Since the training and validation sets are writer-independent, the validation set indicates the model's ability to generalize to unseen writer styles. To maximize the available data, no separate test set was used. Data from the IAM-HTR was also not used to prevent selecting a model in favor of it. In addition to the words present in the training set, 22,500 unique words from the Brown corpus [67] were used as text input. This allowed the generator to handle a wider variety of words. The number of training epochs was determined based on the convergence of the losses and the final model was selected based on the quality of synthetic images between the training epochs.

The quality of the synthetic images was evaluated through comparison with real images and the FID (see Equation 5). While there is discourse about the appropriateness of the FID for synthetic images with handwriting (see Section 3.3.4), the metric is the only one consistently used in HTG. Therefore, the FID was used for model selection. Specifically, the FID was computed every 250 epochs starting from when the losses began converging.

Another method that indicates image quality is the comparison of the performance of an HTR architecture on real data when it is trained separately on a real and recreated training set [5]. If the

synthetic images are similar to the real images, the performance of the two models on real test data should be similar. Evaluating GANwriting in this manner during training is infeasible, however, as recreating a dataset such as IAM-GEN is time-consuming. Therefore, this method is only applied to the final model. Specifically, it was applied to the IAM-HTR dataset since experiments for image-based self-supervised HTR already require it to be recreated.

4.4 Self-supervised Handwritten Text Recognition

4.4.1 Model Integration

When making decisions about data preprocessing and model architectures in the previous sections, the integration of HTR and HTG into the self-supervised framework was carefully considered. To reiterate, we proposed to incorporate a pretrained generator into a self-supervised HTR framework by directly inputting the predicted character probabilities to this generator, and computing the loss between the resulting synthetic image, and the HTR's input image. Moreover, the synthetic and input images' texts should have the same writing style. For the HTR architecture, the best-performing supervised model was used. For the HTG architecture, GANwriting was used.

Following the methodology for image-based self-supervised HCR in Section 3.4, the image preprocessing and text label encoding were identical between the HTG and HTR pipelines, and GANwriting was conditioned on one-hot encoded text labels. Unlike the HCG's generator, GANwriting's generator was also conditioned on writer style. Hence, for self-supervised HTR there was an additional question of how to ensure that the synthetic and input images have the same writer styles. Considering that the style encoder requires a set of K style images, there were two options: 1) duplicate the input image K times, or 2) use writer labels to randomly select K word images from the same writer, with replacement. As the second method resulted in a 'higher quality of synthetic images, we opted for this method. The next section discusses the last key component to training the self-supervised framework: the loss function.

4.4.2 Loss Functions

Through the experiments for image-based self-supervised HCR, we found that high-level loss functions are necessary for good model performance. Therefore, only higher-level loss functions were considered for self-supervised HTR. Specifically, the perceptual loss was investigated further after it showed potential for self-supervised HCR. Additionally, experiments were conducted with the Structural Similarity Index Measure (SSIM), which has been employed as an image quality evaluation metric as well as a loss function for image reconstruction tasks [68].

Results for GANwriting showed that it struggles to imitate the writer's styles, resulting in stylistic differences between real and synthetic images. These include differences in slant, character shape, and scale. Preliminary experiments suggested this is detrimental even when using higher-level loss functions such as the perceptual loss. Therefore, we propose two style-invariant losses in addition to the image-based perceptual and SSIM loss functions. These are a Siamese Network loss, inspired by the field of word spotting, and a text-based content loss. The following details each of the four loss functions.

Perceptual Loss [61] This loss function was considered as it resulted in decent performance for self-supervised HCR. However, this does not guarantee similar results for self-supervised HTR, where the data has a considerably higher variability. Following the experiments for self-

supervised HCR, the perceptual loss was computed with the activations of feature maps extracted at the 7th convolution. The perceptual loss is given by Equation 8.

Structural Similarity Index Measure The SSIM is a metric commonly used to evaluate image quality in image generation tasks, including that of HTG. It has also shown to be useful in image reconstruction tasks [68], which is to an extent similar to the current task, where an HTR network learns to predict character sequences such that the synthetic images generated with these are similar to the input images. The SSIM compares images w.r.t. their luminance, contrast, and structure. It is computed per pixel with a sliding symmetric Gaussian window. Let p and $\hat{p}$ be pixels in a pair of input and synthetic images, such that p and $\hat{p}$ are at the center of an image patch defined by the sliding window. The SSIM is then computed as in Equation 16, where μ and σ represent the mean and standard deviations of the image patches, and $\sigma_{\hat{p}p}$ the covariance between the patches of the input and synthetic images. C_1 and C_2 are constants. The SSIM ranges from -1 to 1, where 1 indicates perfectly similar images. The SSIM loss between a pair of input and synthetic images (y, $\hat{y}$) is then given by Equation 17, where P is the number of pixels.

$$SSIM(p,\hat{p}) = \frac{(2\mu_{\hat{p}}\mu_p + C_1)(2\sigma_{\hat{p}p} + C_2)}{(\mu_{\hat{p}}^2 + \mu p^2 + C_2)(\sigma_{\hat{p}}^2 + \sigma_p^2 + C_2)} \tag{16}$$

$$L_{SSIM}(y,\hat{y}) = 1 - \frac{1}{P}\sum_{i=1}^{P} SSIM(y_i,\hat{y}_i) \tag{17}$$

Siamese Network Loss One method to circumvent stylistic differences and focus on textual content is to train a Siamese Network on same- and different-word images. This was inspired by Barakat et al. [69], where a Siamese Network was successfully employed for word spotting. Such a network learns to extract feature representations such that a distance metric is small for images belonging to the same class, and large for images belonging to different classes. It does this by mapping pairs of input images to their respective feature representations, using the same network for both inputs. Applied to word images, images with the same text are considered to belong to the same class, and to a different class otherwise. In this manner, we aimed to circumvent the stylistic differences between the synthetic and real images.

Given a pretrained Siamese Network ϕ that is able to extract distinct feature representations that are similar when real and synthetic images contain the same words and dissimilar when they contain different words, the Siamese Network loss is computed as in Equation 18, where y and $\hat{y}$ are real and synthetic images, respectively.

$$L_{sia} = ||\phi(\hat{y}) - \phi(y)||_2^2 \tag{18}$$

The Siamese Network employed for the loss computation was trained on a refactored IAM-GEN dataset (IAM-GEN-SIA) that contains all writer styles in its training, validation, and test sets. Moreover, for each real image in this dataset, two similar and two dissimilar real and synthetic image pairs were generated with GANwriting's best-performing generator. The Siamese Network was trained with the contrastive loss function, where the Euclidean distance was used as the distance function. The network was then evaluated by computing the Euclidean distance between the similar and dissimilar image pairs on the validation and test sets. Comparing the distances between similar pairs ($\mu = 0.107$, $\sigma = 0.07$) and dissimilar pairs ($\mu = 0.539$, $\sigma = 0.057$)

on the IAM-GEN-SIA test set, a two-tailed t-test indicated that the similar pairs had a statistically significant smaller mean distance ($t(2282) = 161.024$, $p < 0.005$). Therefore, it could be expected for the Siamese Network loss on the IAM-HTR dataset to indicate whether a real image and its corresponding synthetic image are similar on IAM-HTR. The methodology and results of the Siamese Network are further detailed in Appendix A

Content Loss While the Siamese Network loss was designed to circumvent the stylistic differences between real and synthetic images, its potential weakness is that it does not explicitly encode the word images' texts. Hence, we also introduced a style-invariant text-based content loss that explicitly encodes the word images' texts. The idea behind this content loss is similar to the Siamese Network loss, where instead of using a pretrained Siamese Network, a pretrained HTR model is applied to the real and synthetic images. Hence, the formula is the same as Equation 18, where ϕ now represents the pretrained HTR network.

For the pretrained HTR network, the experimental setup of the best-performing network of the experiments conducted in the supervised setting was used. The network was pretrained on the IAM-GEN-SIA dataset to ensure independence with the self-supervised HTR model. Its performance on the IAM-GEN-SIA training and test set are shown in Table 6, and its performance in comparison to the best-performing HTR model trained on IAM-HTR is shown in Table 7. Its increase in performance, depicted by an overall decrease in CER and WER, indicates the IAM-GEN-SIA dataset may have contained higher-quality images. Important to note is that the self-supervised HTR model's performance could only be as good as the pretrained HTR model's performance on the IAM-HTR dataset, with the pretrained network acting as a bottleneck.

Table 6: Performance of the best-performing supervised HTR model trained and evaluated on the real images of IAM-GEN-SIA.

	Validation set		**Test set**	
Training Data	CER (%)	WER (%)	CER (%)	WER (%)
IAM-GEN-SIA	11.34	28.00	12.09	28.96

Table 7: Performance of the best-performing supervised HTR model trained on the real images of IAM-GEN-SIA, evaluated on the IAM-HTR validation sets.

	IV-Validation		**OOV-Validation**	
Training Data	CER (%)	WER (%)	CER (%)	WER (%)
IAM-GEN-SIA	**10.89**	**26.38**	**24.39**	**63.39**
IAM-HTR	15.14	32.91	45.57	92.84

4.4.3 Experimental Setup

The self-supervised HTR system was trained on the IAM-HTR-IV training dataset and evaluated with the IAM-HTR-IV and -OOV validation and test sets. To answer the primary research question of whether the proposed self-supervised framework can be effectively used for HTR, experiments were conducted where the system was trained from scratch. Specifically, the HTR network weights were

initialized with the He uniform initialization scheme for more stable training [70]. The models were trained with an initial learning rate of 0.001, which was determined based on preliminary experiments, where larger or smaller learning rates did not result in the learning of relevant information.

Next to training from scratch, transfer learning was used with the best-performing supervised HTR network, trained on IAM-GEN-SIA. This was the same HTR model as was used for the text-based content loss. With experiments in this pretrained self-supervised setting, we aimed to address the research question of whether the proposed self-supervised HTR framework can be used to improve existing HTR models. Since the supervised HTR model was trained with an initial learning rate of 0.0001, this was also used as an initial learning rate for experiments in the pretrained setting.

The experiments in the 'training from scratch' and pretrained settings were conducted for each of the perceptual, SSIM, Siamese Network, and text-based content loss functions to investigate the optimal loss function for the proposed self-supervised HTR framework. Since the HTR model in the pretrained setting was the same as that used for the text-based content loss, it is unlikely to show improvements in this setting. To confirm this, however, the content loss was still included in the experiments in the pretrained setting. Considering then both the training from scratch and pretrained settings, a total of 8 experiments were performed.

When the above experiments did not lead to promising results, especially for the image-based losses, it was considered to conduct the experiments in the training from scratch and pretrained settings on the IAM-HTR dataset recreated with synthetic images. By minimizing the stylistic differences between input and synthetic images in this manner, it could be investigated whether the low model performances were caused by the pretrained generator not accurately imitating the writer's styles. Since these experiments focus on the style, they were only performed with the two image-based losses. Therefore, an additional four experiments were conducted. Based on preliminary experiments, the models in the pretrained setting were trained with an initial learning rate of 0.0003 instead of 0.0001 (see Appendix C). This learning rate was also considered for the pretrained setting on real data, but this was not found to be beneficial (see Appendix B). To explicitly show whether the self-supervised HTR framework can be used to learn new words, the experimental settings of the best-performing models were also used to train the self-supervised models on the IAM-HTR-OOV dataset. This is further addressed in Appendix D.

All models were trained with the Adam optimizer. As aforementioned, when training from scratch, the initial learning rate was 0.001. In the pretrained setting, this was 0.0001, and 0.0003 on the real and recreated IAM-HTR datasets, respectively. Moreover, when the IAM-HTR-IV validation loss did not decrease for more than 10 epochs, the learning rate was reduced by a factor of 0.5. The batch size was set to 16. Additionally, to prevent overfitting, early stopping was applied when the IAM-HTR-IV validation loss did not decrease for more than 15 epochs.

4.4.4 Evaluation

Following the evaluation of the supervised HTR models, the self-supervised HTR models were evaluated with the CER and WER. The performances of the self-supervised HTR models could thus also be compared to the best-performing supervised model trained on the IAM-HTR set. Additionally, the loss curves were considered to indicate whether learning occurred, and a set of synthetic images with HTR-predicted text was analyzed to obtain a deeper understanding of the results and dynamics of the self-supervised framework.

5 Results

This Chapter reports the results of the experiments described in Chapter 4. Based on the results of experiments conducted for supervised HTR, an HTR architecture was selected for the self-supervised HTR framework in Section 5.1. Likewise, in Section 5.2, the best-performing pretrained generator was selected and used in the self-supervised HTR system. Lastly, Section 5.3 reports the results of the experiments performed with the image-based and style-invariant losses for self-supervised HTR.

5.1 Supervised Handwritten Text Recognition

To determine the optimal HTR model architecture and experimental setup, HTR models were trained in a supervised setting on the IAM-HTR-IV training set and evaluated with the IAM-HTR-IV and -OOV validation and unseen test sets. The best-performing model architecture and experimental setup were then used in the self-supervised HTR experiments. Supervised models with Puigcerver's [16] CNN-BLSTM architecture were trained where in one setting, column-wise concatenation was used to reduce the CNN's feature maps to one-dimensional vectors, and in a second setting, column-wise max-pooling was used. Performances of the CTC and CE loss functions were also compared, resulting in four models. To contextualize this model's performance, a comparison is made with the literature. The best-performing model was selected based on the CER and WER on the validation sets to prevent tuning on the test set.

Table 8 displays the results on the IAM-HTR validation sets. It shows that column-wise max pooling led to an increased performance compared to column-wise concatenation for both the CE and CTC losses, indicated by a decrease in CER and WER for both IV and OOV validation sets. For the HTR trained with the CE loss, this decrease was strongest with a 56.37% reduction in CER and a 40.61% reduction in WER on IV validation data. This indicates that HTR models benefit from extracting features that focus on the characters' horizontal position, as is the result of column-wise max pooling. Given these results, we selected the Puigcerver architecture adapted with column-wise max pooling as the best-performing architecture.

Considering the loss functions, the results overall showed a decreased CER and WER on IV validation data for the CE loss compared to the CTC loss, and a decreased WER on OOV validation data. Only the OOV CER of the CE loss was increased compared to that of the CTC loss across experimental conditions. Since correctly predicted words are preferred over correctly predicted individual characters, the WER was prioritized. Taking into account both the architecture and loss function, it can be determined that the best-performing model used column-wise max pooling in combination with the CE loss. This is further confirmed when comparing the CER and WER of models trained with the CE and CTC losses in combination with column-wise max pooling on the test set (Table 9).

Table 8: Validation CER and WER on IAM-HTR per supervised HTR model configuration.

	IV Validation		**OOV Validation**	
Method	CER (%)	WER (%)	CER (%)	WER (%)
CE + concatenation	34.70	55.42	68.89	99.42
CE + max pooling	**15.14**	**32.91**	45.57	**92.84**
CTC + concatenation	31.28	58.22	65.54	99.88
CTC + max pooling	20.65	51.86	**43.66**	95.84

In Table 10, the best-performing model's results on the IAM-HTR-IV test set are compared to those achieved by word-level HTR models in the literature. It shows that our model performs substantially worse compared to previous research. It has to be noted, however, that our model has been trained on roughly half the amount of data compared to models proposed in the literature. Additionally, this comparison only serves to contextualize the performance of the adapted Puigcerver model on the word level, as the results across models are reported for different test partitions of the IAM dataset.

Next to the replacement of column-wise concatenation and the CTC loss with the CE loss, the results also inform us about the generalizability of the HTR models to unseen data. Specifically, the validation and test results both show a substantial increase in CER and WER, confirming the suggestion made in HTR literature that handwritten text recognizers struggle to generalize to new words. In fact, above 90% of the OOV words were transcribed incorrectly in both OOV validation and test sets by all supervised models.

Table 9: Test CER and WER on IAM-HTR per supervised HTR model configuration.

	IV Test		**OOV Test**	
Method	CER (%)	WER (%)	CER (%)	WER (%)
CE + max pooling	**14.71**	**30.89**	46.67	**93.13**
CTC + max pooling	20.40	51.07	**44.28**	96.48

Table 10: Word-level HTR results in the literature on the IAM database

Approach	**Method**	**CER (%)**	**WER (%)**
Nearest neighbour	PHOC [20]	11.27	20.01
	Deep PHOC [21]	3.72	6.69
	HWNET v3 [23]	**1.67**	**3.62**
Object recognition	Mondal et al. [24]	9.53	29.21
Attention-based	WordStylist Real IAM [5]	4.86	14.11
	CNN-BGRU [28]	6.88	17.45
CRNN	CE + max pooling ([16], adapted)	14.71	30.89

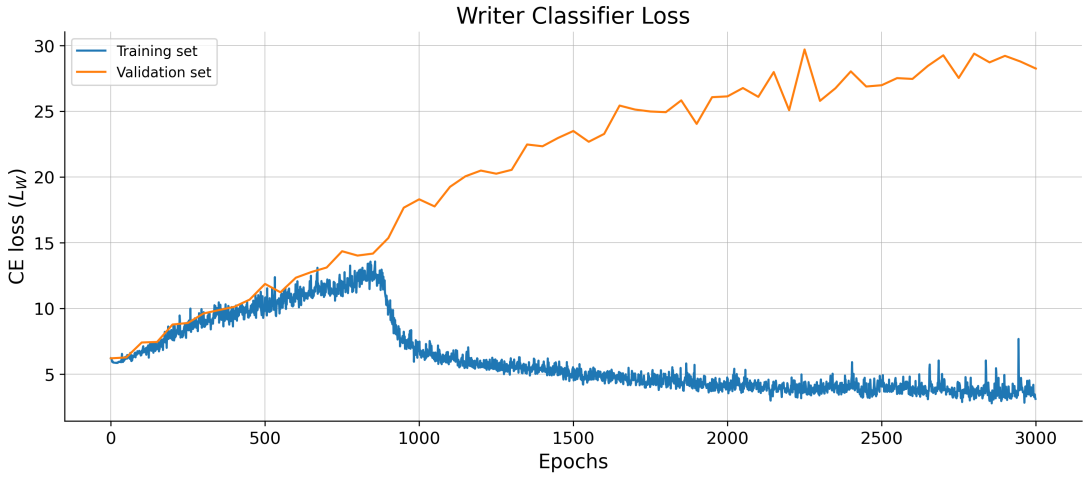


Figure 33: GANwriting writer classifier loss curve

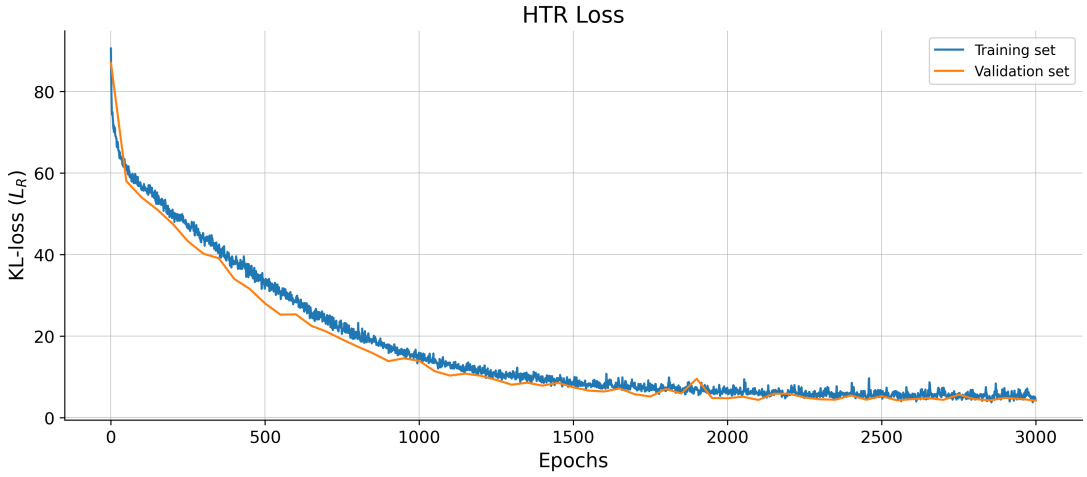


Figure 34: GANwriting HTR loss curve

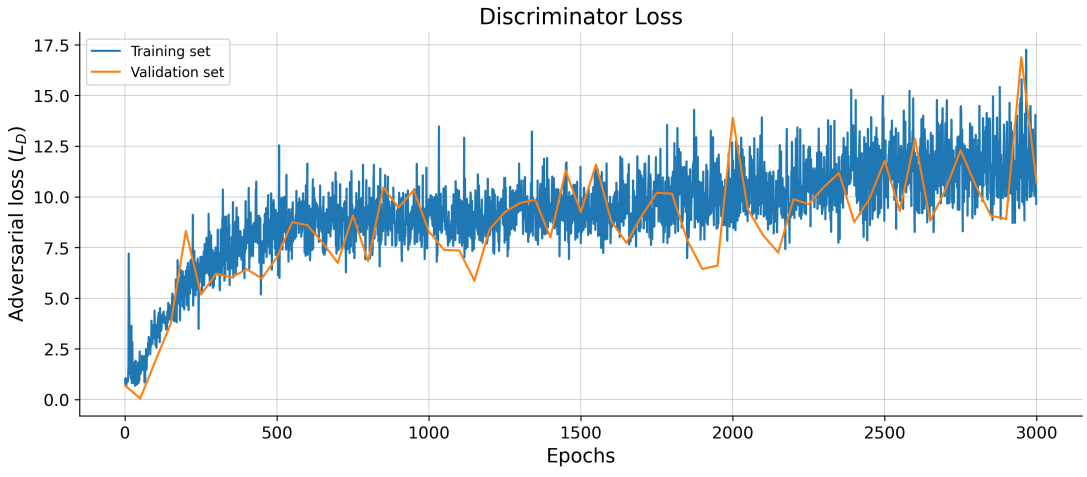


Figure 35: GANwriting discriminator loss curve

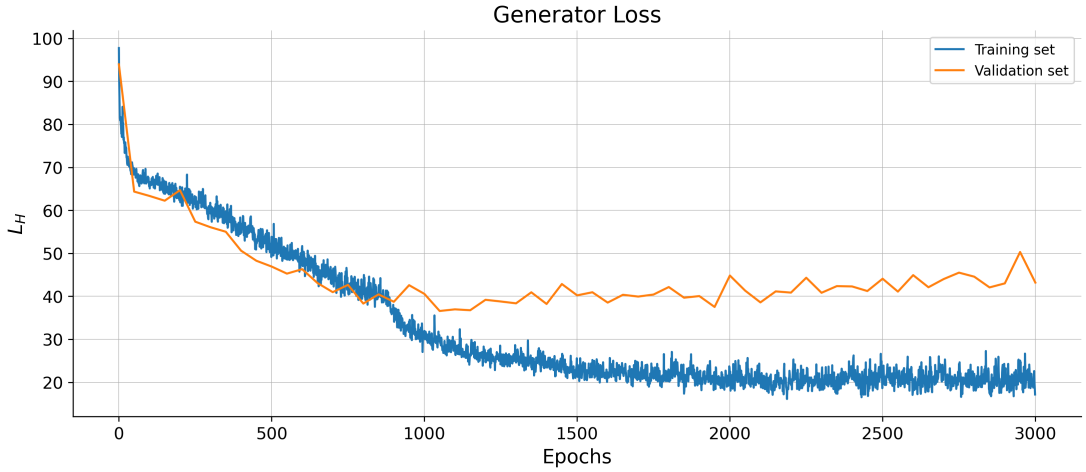


Figure 36: GANwriting generator loss curve

5.2 Handwritten Text Generation

The generative model GANwriting [39] was trained on the IAM-GEN dataset as portrayed in Figure 29. GANwriting was trained end-to-end, and evaluated with the FID on the validation set. Unfortunately, the losses of the text recognizer, writer classifier, and discriminator models were saved only on the synthetic images and not for their own training processes on real images. Though this limits the analysis of the loss curves, they can still inform us about the system's overall stability and convergence. Considering this, the system was trained for 3000 epochs as the generator's validation loss had converged at this point. Since the FID calculations involve recreating the IAM-GEN validation set, the FID was computed only from epoch 1750, when the generator loss started to converge. Lastly, we also compare the performance of the best-performing supervised HTR model on real validation data, when it is trained on the IAM-HTR dataset with real images, and on IAM-HTR recreated with the best-performing generator model.

The loss curves for the writer classifier, text recognizer, discriminator, and generator models of GANwriting on synthetic images are shown in Figures 33, 34, 35, and 36, respectively. Noticeable is the writer classifier's increasing validation, which is likely due to the training and validation sets being writer-independent. The training loss indicates that from the 1000th epoch, the generator started to better imitate the writer's styles. These observations are also reflected in the generator's loss. The other loss curves do not display unexpected behavior during training: the HTR and generator losses decrease, while the discriminator losses increase.

Figure 37 displays a set of 20 synthetic images across epochs and their real counterparts. Overall, a discrepancy can be observed between the writers' styles in the real and synthetic images. The model appeared to struggle with slanted writer styles and produced mostly loose characters. Moreover, most synthetic word images were on a different scale as their corresponding real word image. Considering legibility, the generator struggled with creating the letters 'w' and 'm', as well as the combination 'ng' in this set of images. Additionally, the all-capital word 'THERE' (fourth row) does not have clearly distinguishable characters in the synthesized images. At epoch 3000, artifacts are sometimes present as an ink blob in the upper right of an image. It is difficult to draw any conclusions about which model produces higher quality images based on this set of images, as it is only a small set.

The FID scores in Table 11 provide a more general and objective comparison of the image quality of the synthesized images across epochs, where a lower FID indicates an increased synthetic image quality. As such, the generator at epoch 3000 resulted in the best synthetic image quality (FID =

Table 11: FID scores across epochs of GANwriting on IAM-GEN validation data.

Epoch	1750	2000	2250	2500	2750	3000
FID	46.12	40.96	42.93	38.08	30.65	**30.36**

30.36), and the worst at epoch 1750 (FID = 46.12). Based on these results, we conclude that the generator model at epoch 3000 provides images of better quality compared to the models at previous epochs, in terms of the FID. Therefore, this model was used for the HTR experiments in the self-supervised setting.

To obtain a more thorough impression of the synthetic image quality, the best-performing supervised HTR model (using column-wise max pooling and the CE loss) was trained on the IAM-HTR dataset with real images (real IAM-HTR), and on the IAM-HTR dataset recreated with the best-performing generator (recreated IAM-HTR). We compared only the validation results, as these results influenced the decision to use style-invariant losses for self-supervised HTR. Table 12 then shows the results of the supervised HTR model trained on the recreated IAM-HTR dataset, and evaluated on the real and recreated IAM-HTR validation sets. It shows the performance of the HTR model does not generalize well to the real IAM-HTR validation data. Additionally, Table 13 shows the performances of the HTR models evaluated on the real IAM-HTR validation datasets when they are trained on either the real IAM-HTR or the recreated IAM-HTR dataset. It shows that the supervised HTR model performed overall worse when trained on the synthetic images compared to the real images. Only on the OOV validation data did the model trained on synthetic images perform slightly better compared to real images, as indicated by the decreased WER. Overall, these results indicate that there is a discrepancy between the real and synthetic images due to the aforementioned differences in style and legibility.

Table 12: Real and Recreated IAM-HTR validation results of the supervised HTR model (using CE and column-wise max pooling) trained with the recreated IAM-HTR dataset.

	IV		**OOV**	
Validation Data	CER (%)	WER (%)	CER (%)	WER (%)
Recreated IAM-HTR	0.67	2.91	5.25	21.59
Real IAM-HTR	53.43	82.82	57.21	95.50

Table 13: Real IAM-HTR validation results of the supervised HTR model (using CE and column-wise max pooling) trained on the real and recreated IAM-HTR dataset.

	IV Validation		**OOV Validations**	
Training Data	CER (%)	WER (%)	CER (%)	WER (%)
Recreated IAM-HTR	**53.43**	**82.82**	**57.21**	**95.50**
Real IAM-HTR	20.65	51.86	43.66	95.84

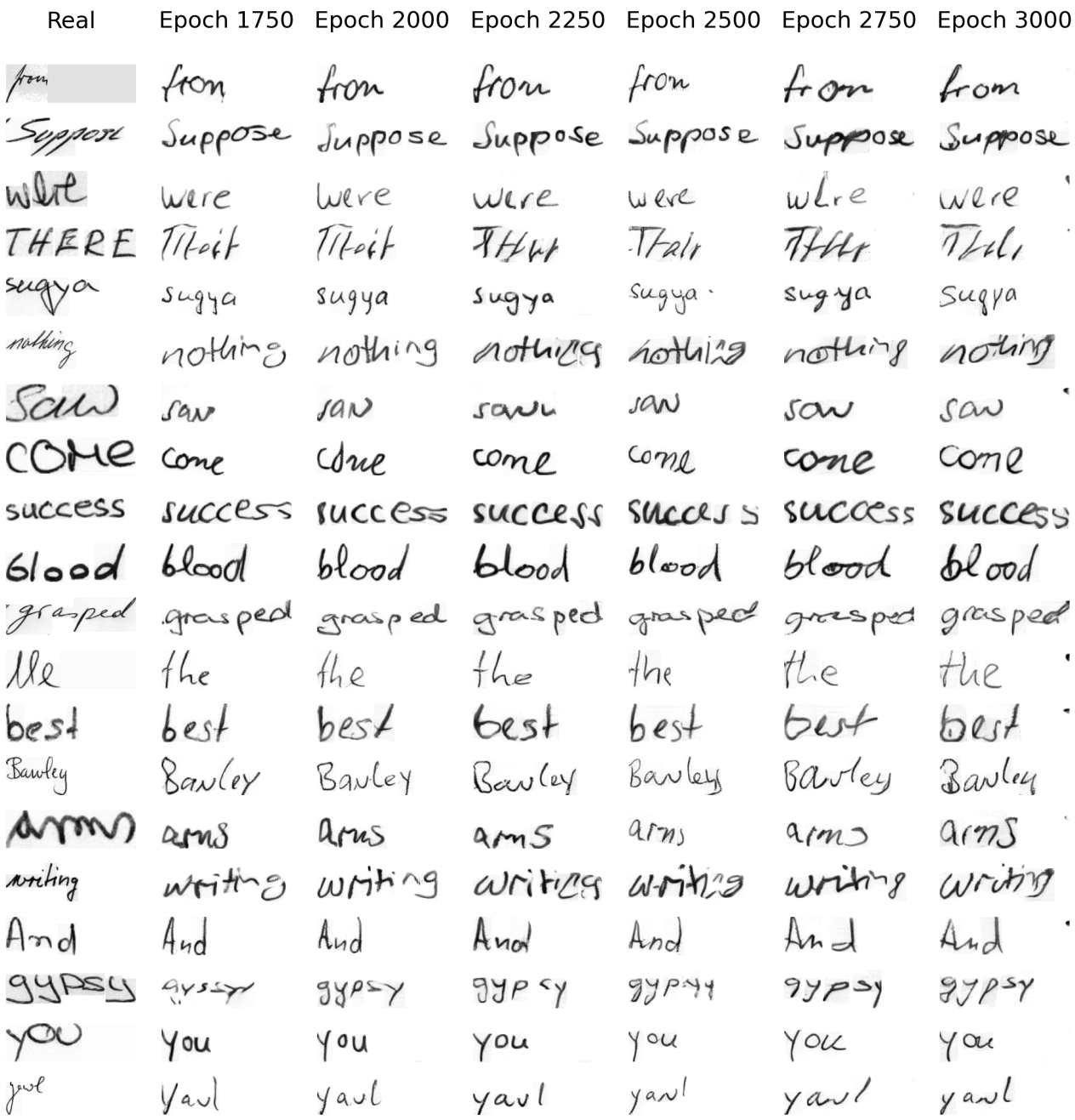


Figure 37: Generated images across epochs 1750 to 3000 in steps of 250 epochs.

5.3 Self-supervised Handwritten Text Recognition

A total of 12 experiments were conducted to investigate self-supervised HTR. Here, we used the best-performing HTR architecture, which leveraged column-wise max pooling, and the best-performing generator, which was trained for 3000 epochs. A total of four loss functions were investigated, of which two were the image-based perceptual and SSIM loss functions, and two were the style-invariant Siamese Network and text-based content loss functions. The models were trained on the HTR-IAM-IV training data and evaluated with the IAM-HTR-IV and -OOV validation and test sets. This was done with real data, and separately with the IAM-HTR dataset recreated using the pretrained generator. The models' weights were initialized with He uniform weight initialization (from scratch), or with the best-performing supervised HTR model pretrained on the IAM-GEN-SIA dataset. All models were evaluated with the CER and WER. In addition, model behavior was observed through loss curves and a set of synthetic images containing predicted texts. We first report the results of the experiments conducted on real data for the image-based and style-invariant losses, separately, and then the results of the experiments on the recreated dataset.

5.3.1 Results on Real IAM-HTR Data

Table 14 shows the results of the self-supervised models on the real IAM-HTR dataset.

Image-based Losses When considering the image-based losses, both the SSIM and perceptual losses showed extremely high error rates, with a WER of 100% in all training settings (from scratch and pretrained), meaning not a single word was predicted correctly. For the models trained from scratch, however, it cannot be said no learning occurred, as the loss curves in this setting for both SSIM and perceptual losses decreased (see Figures 38 and 39, respectively). For the self-supervised models in the pretrained setting, the high error rates indicate catastrophic forgetting occurred, likely in the first epoch, as the corresponding loss curves (for both loss functions) have similar starting points as the models when training from scratch. Then, it can be seen that the self-supervised systems trained with the SSIM and perceptual losses converge to relatively high values, though the minimum for both is 0.0.

Table 14: Self-supervised HTR results on the real IAM-HTR validation sets.

		IV-Validation		**OOV-Validation**	
Loss Function	**Pretrained**	CER (%)	WER (%)	CER (%)	WER (%)
SSIM	No	96.33	100	94.14	100
	Yes	95.64	100	95.06	100
Perceptual	No	93.63	100	93.68	100
	Yes	94.49	100	94.22	100
Siamese Network	No	94.03	100	93.87	100
	Yes	59.89	81.98	74.94	99.89
Content	No	97.07	100	96.10	100
	Yes	21.26	48.50	39.66	88.86
CE (pretrained HTR)	-	**10.89**	**26.38**	24.39	**63.39**

To obtain a deeper understanding of these low performances and model behavior, we turn to Figures 42 and 43. These depict pairs of real input images and synthetic images containing the predicted texts for the SSIM and perceptual losses, respectively. In the two Figures, for both training from scratch from the pretrained models, the synthetic images are relatively similar, with ‘i‘- or ‘j‘-like characters and similar artifacts. Moreover, it can be seen that the predicted characters cannot be discerned in the synthetic images.

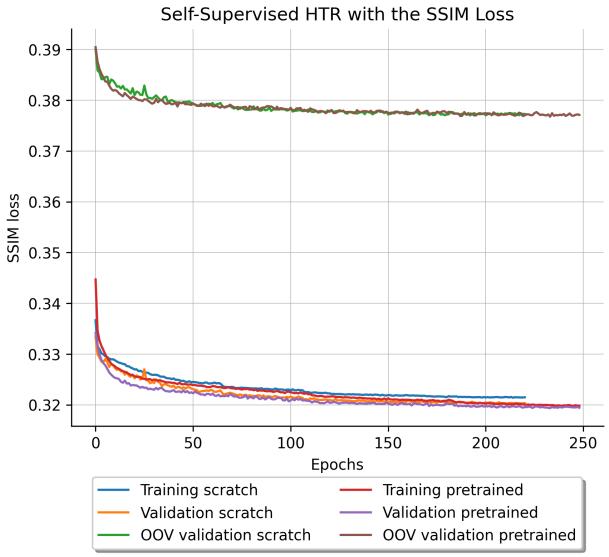


Figure 38: Self-supervised HTR trained with the SSIM loss on real IAM-HTR data loss curve

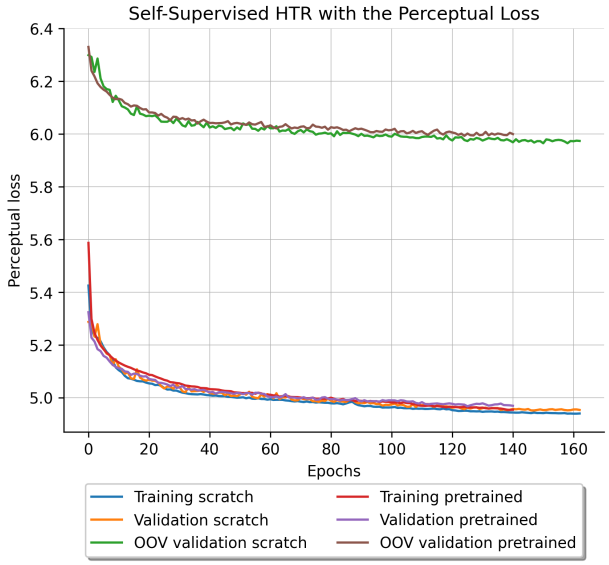


Figure 39: Loss curve of self-supervised HTR trained with the perceptual loss on real IAM-HTR data.

Style-invariant losses The results for the style-invariant losses when the self-supervised models were trained from scratch show, like the image-based losses, high error rates. Hence, while the loss curves for the models trained from scratch with the Siamese Network and content losses show a decreasing trend (see Figures 40 and 41, respectively), the information learned was not useful for the HTR task. When the models were trained in the pretrained setting, the error rates were substantially lower compared to training from scratch, specifically for the content loss. However, when compared to the performance of the pretrained supervised HTR model, it can be seen that information was not learned, but forgotten. This is reflected in the loss curves corresponding to the pretrained setting, where an initial increase can be observed for both style-invariant loss functions. Due to the early stopping that was applied, it is unclear whether the proposed loss functions could have led to improved performances.

The loss curves do not explain all differences in performance, however. For the Siamse Network loss, for example, the models trained from scratch and in the pretrained setting have similar final loss values (on both IV and OOV data), even though the latter showed a substantially lower CER and WER compared to the models trained from scratch. This can also be observed in the performance of the text-based content loss on the HTR-IAM-OOV validation data in the pretrained setting compared to the IAM-HTR-IV training and validation data when training from scratch. This suggests the objective functions may not appropriately capture meaningful differences between two images w.r.t. text.

Considering the pairs of real and synthetic images with predicted texts for the Siamese Network

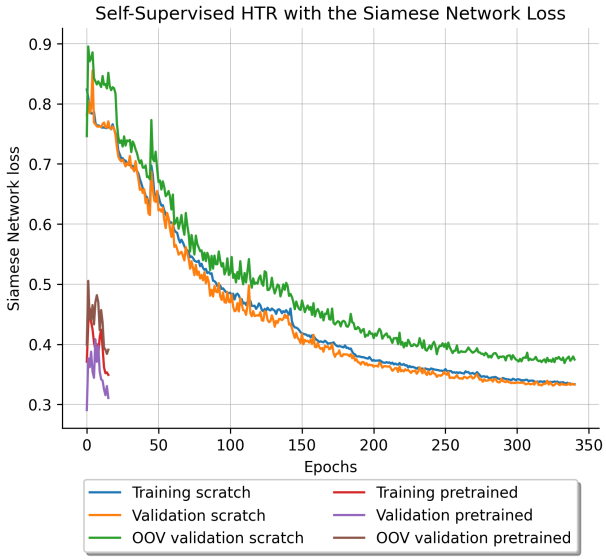


Figure 40: Loss curve of self-supervised HTR trained with Siamese Network loss on real IAM-HTR data.

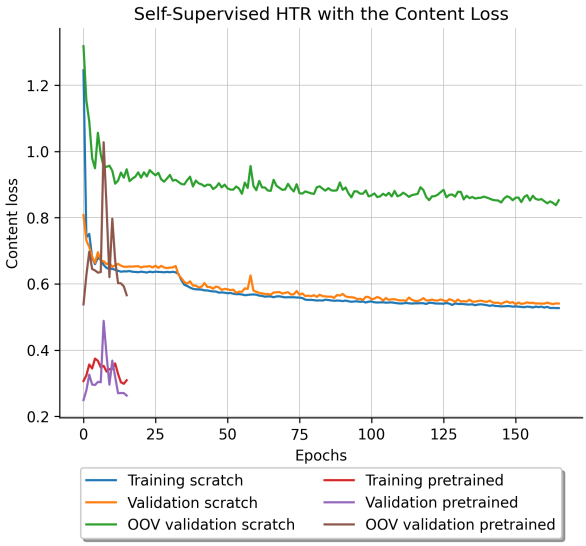
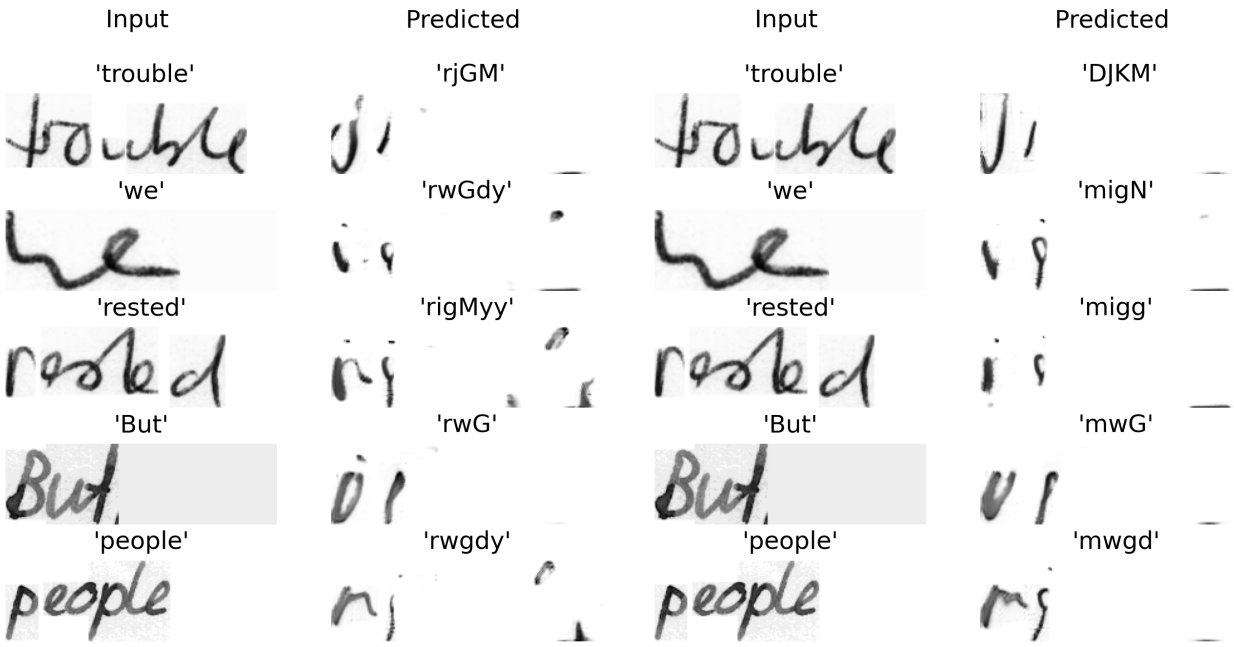


Figure 41: Loss curve of self-supervised HTR trained with the text-based content loss on real IAM-HTR data.

loss in Figures 40, it can be seen that when the model was trained from scratch, there are more characters present, but they are not necessarily legible. Some characters match those in the predicted text, but overall the words do not match. Likely, the self-supervised HTR network exploits weaknesses in the pretrained Siamese Network to minimize the loss in this manner. In the pretrained setting, the predicted words are generally evident in the synthetic images, but the predictions were still mostly incorrect.

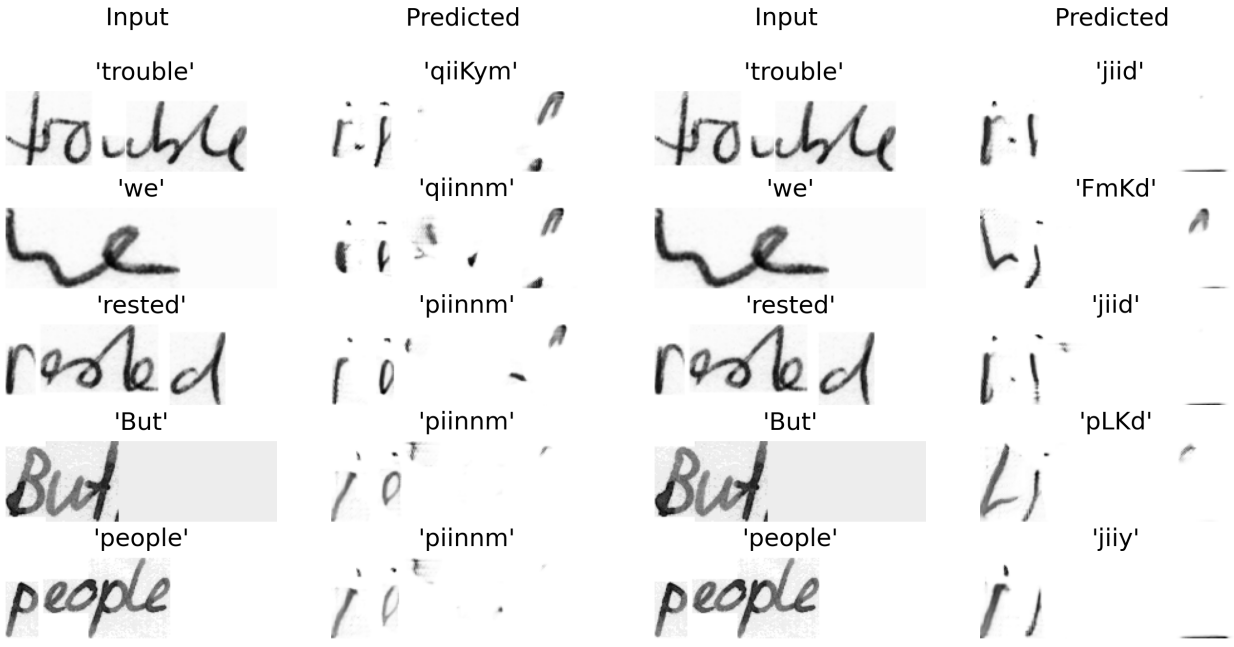
For the text-based content loss, the image pairs for the self-supervised HTR models trained from scratch and from the pretrained supervised HTR models are shown in Figures 45a and 45b, respectively. When training from scratch, it can be seen that, in the given set of images, the synthetic images are similar to each other. This suggests that the self-supervised HTR network learned to generate a certain pattern that minimizes the content loss sub-optimally; the images clearly show no important information was learned. For the image pairs in the pretrained setting, on the other hand, the predicted characters are clearly present.

Comparing the Siamese Network loss with the text-based content loss given the above results, the explicitly encoded texts led to increased performance when leveraging background knowledge of the supervised HTR model. This is likely due to the fact that the content loss leveraged the same supervised HTR network as used for transfer learning. It could be that training this self-supervised model with the text-based content loss for a longer time may result in the forgetting of more information. When training the models from scratch, the images produced with the Siamese Network loss's self-supervised HTR model appeared more realistic. Still, neither loss functions resulted in lexical, nor legible words.



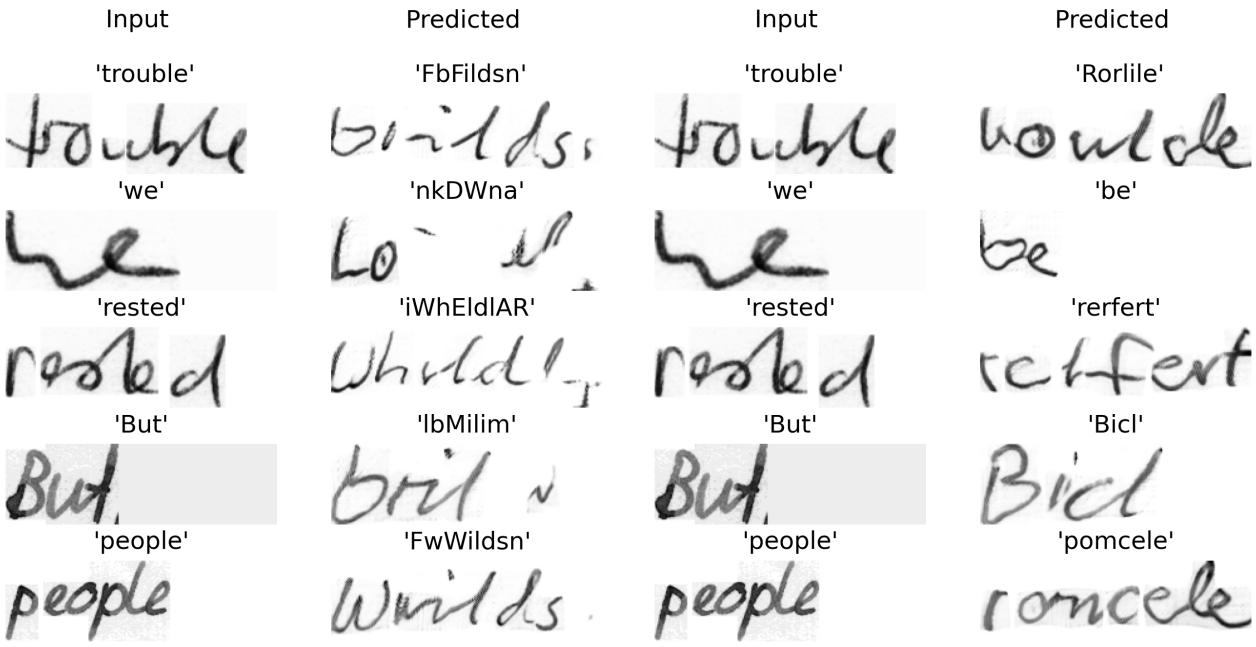
(a) From scratch (b) Pretrained

Figure 42: Input images and their corresponding synthetic images with the predicted text for the SSIM loss trained (a) from scratch and (b) from the pretrained supervised HTR model.



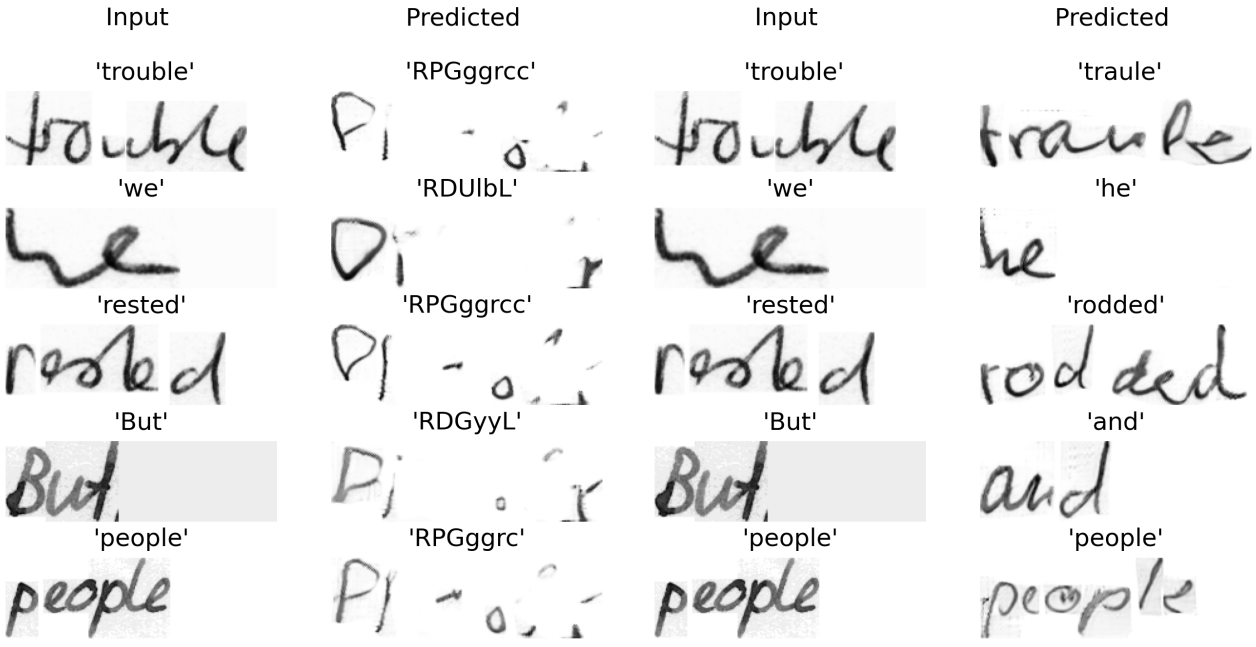
(a) From scratch (b) Pretrained

Figure 43: Input images and their corresponding synthetic images with the predicted text for the perceptual loss trained (a) from scratch and (b) from the pretrained supervised HTR model.



(a) From scratch (b) Pretrained

Figure 44: Input images and their corresponding synthetic images with the predicted text for the Siamese Network loss trained (a) from scratch and (b) from the pretrained supervised HTR model.



(a) From scratch (b) Pretrained

Figure 45: Input images and their corresponding synthetic images with the predicted text for the text-based content loss trained (a) from scratch and (b) from the pretrained supervised HTR model.

5.3.2 Results on Recreated IAM-HTR Data

The style-invariant losses were designed to investigate whether the stylistic differences between the real and synthetic images were too great for considerable learning to have occurred with the image-based losses. Given the above results on real images, it could be questioned whether these stylistic differences are the real cause of the low performances. This was further investigated through experiments where the self-supervised HTR framework was trained with the image-based losses on the recreated IAM-HTR dataset.

Table 15 shows the results of the self-supervised HTR framework trained with the image-based SSIM and perceptual loss functions on the recreated IAM-HTR dataset. Considering the models trained from scratch, the results indicate only limited learning of relevant information occurred as a result of minimizing the stylistic differences between input and synthetic images. Still, this is an improvement in comparison to the model performances of the image-based losses on real training images. The more promising results for the image-based losses with synthetic images, however, are of the model performances in the pretrained setting. For both image-based loss functions, the overall decreased CER and WER on the recreated IAM-HTR-IV and -OOV validation data in the pretrained setting compared to the pretrained supervised HTR model indicates new information was learned by training on synthetic images. These findings are reflected in the loss curves for the SSIM and perceptual loss functions depicted in Figures 46 and 47, respectively. Overall, these results on recreated IAM-HTR data suggest that the image-based self-supervised models trained with real images performed badly partly due to the stylistic differences between the real and synthetic images.

To obtain more insight into the image-based self-supervised HTR models on the recreated IAM-HTR dataset, we inspected the set of image pairs with input and synthetic images (containing the predicted text) in Figures 48 and 49 for the SSIM and perceptual loss functions, respectively. For both loss functions in the pretrained settings (Figures 48b and 49b), the predicted texts were clearly legible in the synthetic images, showing that the models indeed learned relevant information. Regarding the image-based models trained from scratch, some of the image pairs contain ‘i‘- or ‘j‘-like characters along with artifacts, similar to the image pairs for the image-based losses with real data. Additionally, the image pairs with synthetic data contained some of the predicted characters. Still, these image pairs show the models could not learn information that is necessary for the HTR task from scratch.

Given the results in Table 15, the image-based self-supervised HTR model trained with the perceptual loss in the pretrained setting had the best performance, having the lowest CER and WER on the IAM-HTR-IV and -OOV validation sets. This model was therefore applied to the recreated and real IAM-HTR test sets in order to analyze its generalizability. In addition, we compare it to the best-performing supervised model trained on the real IAM-HTR dataset. Table 16 shows these results. First, it can be observed that the performance of the self-supervised model on recreated IAM-HTR test data is similar to that on the validation sets. Second, the model performance substantially decreases when applied to real IAM-HTR test data. This indicates that the model does not generalize well to real data. However, this can likely be attributed to the quality of synthetic images limiting the information that could be learned. Third, in comparison to the best-performing supervised HTR model trained on real IAM-HTR data, the test results on the real images show that the image-based self-supervised model performed worse for the IAM-HTR-IV test set, but slightly better on the IAM-HTR-OOV test set. This may be due to the background knowledge learned by the supervised HTR model pretrained on IAM-GEN-SIA, where the IAM-HTR-OOV words were not explicitly excluded. Therefore, we consider the supervised HTR model to have a better overall performance, still.

Table 15: Image-based self-supervised HTR results on the recreated IAM-HTR validation datasets.

		IV Validation		OOV Validation	
Loss function	**Pretrained**	CER (%)	WER (%)	CER (%)	WER (%)
SSIM	No	66.79	85.12	87.99	100
	Yes	0.70	2.71	2.28	9.32
Perceptual	No	66.21	84.12	86.94	100
	Yes	**0.36**	**1.45**	**1.46**	**6.48**
CE (pretrained HTR)	-	5.60	17.05	14.84	49.65

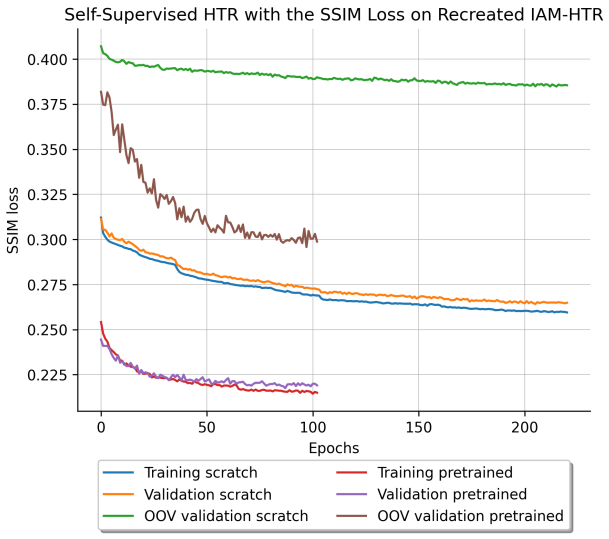


Figure 46: Loss curve of self-supervised HTR trained with SSIM loss on recreated IAM-HTR data.

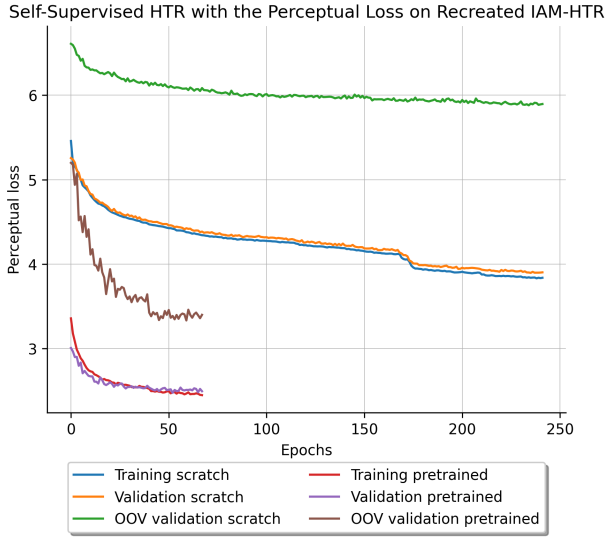
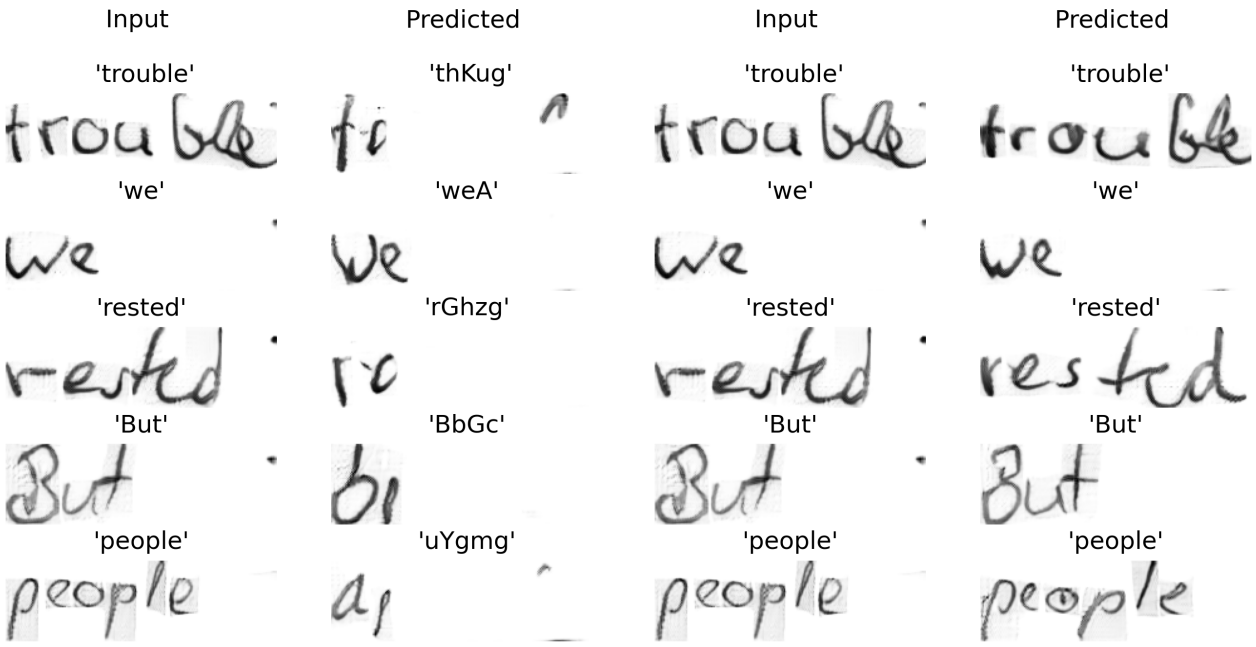


Figure 47: Loss curve of self-supervised HTR trained with perceptual loss on recreated IAM-HTR data.

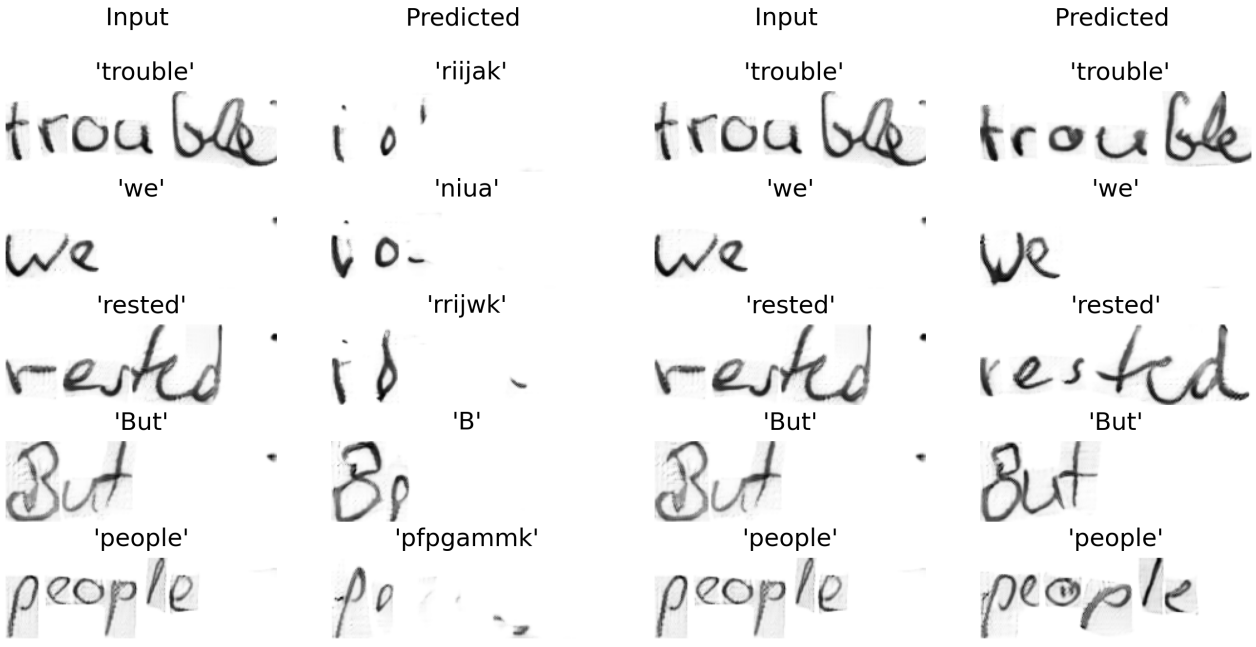
Table 16: Results on real and recreated IAM-HTR test data of the best-performing supervised and image-based self-supervised HTR models, trained on real and recreated IAM-HTR data, respectively. The self-supervised HTR model was trained with the perceptual loss in the pretrained setting.

		IV Test		OOV Test	
Test Data	**Method**	CER (%)	WER (%)	CER (%)	WER (%)
Real IAM-HTR	Image-based self-supervised	36.08	66.40	43.49	87.80
	Supervised	14.71	30.89	44.67	93.13
Recreated IAM-HTR	Image-based self-supervised	0.44	1.56	1.83	7.04



(a) From scratch (b) Pretrained

Figure 48: Input images and their corresponding synthetic images with the predicted text for the SSIM loss trained (a) from scratch and (b) from the pretrained supervised HTR model.



(a) From scratch (b) Pretrained

Figure 49: Input images and their corresponding synthetic images with the predicted text for the perceptual loss trained (a) from scratch and (b) from the pretrained supervised HTR model.

6 Conclusions

This thesis aimed to address the need for vast amounts of annotated data for handwritten text recognition by proposing a novel self-supervised HTR framework relying on the similarity between input and synthetic images, where the synthetic images contained the HTR-predicted text and were produced by pretrained handwritten text generators. The framework is based on the idea that two images written in the same style and with the same text are more similar compared to two images with different texts. This thesis thus addresses the primary question of whether the proposed image-based self-supervised HTR system leveraging HTG can be effectively used for HTR. To investigate this, we asked how such a system can be implemented, what loss functions should be used for training it, and whether it could further improve supervised HTR model performance. First, the feasibility of the framework, its implementation, and potential loss functions were studied on the simpler HCR task with the MNIST dataset. Based on these findings, the research questions were addressed for handwritten word recognition posed as a sequence labeling task with the IAM database.

To implement the image-based self-supervised framework, it was proposed to 1) directly feed the HTR predicted probabilities to the pretrained generator, and 2) apply consistent image preprocessing and label encoding for HTR and HTG models. To enable the latter, the textual contents of synthetic images were controlled by conditioning the generator on one-hot encoded labels. For HCR on the MNIST dataset, experiments were conducted with three different losses operating on pixels of the images or extracted feature maps. Results showed a higher-level loss function is required for decent HCR performance with the proposed implementation. Based on these findings, the proposed implementation of the framework was applied to word recognition. Here, the HTG model GANwriting [39] was trained on half the IAM dataset to produce images with arbitrary writer styles and texts, and integrated with a CNN-BLSTM HTR architecture [16], [64]. Initially, experiments were conducted with two image-based losses, where the self-supervised HTR models were trained on data independent from GANwriting, either from scratch or leveraging transfer learning. When these results were not promising, the possibility of limitations imposed by stylistic differences between real and synthetic images was investigated by proposing two style-invariant losses, and by applying the experiments with image-based losses to data recreated with GANwriting. Only the image-based self-supervised HTR models trained with the recreated data and leveraging transfer learning showed a capability of learning new, relevant information. Still, these models did not generalize well to real data.

Given our findings, let us now first consider the secondary research questions before addressing the primary research question.

RQ1 How can handwritten text generation be effectively incorporated into image-based self-supervised handwritten text recognition?

For the incorporation of HTG into the image-based self-supervised HTR framework, the input and synthetic images needed to be in the same range w.r.t. pixel values, and the character indices needed to represent the same characters in the HTG and HTR models. We found that directly inputting the HTR predictions to the pretrained generator, conditioning the generator on one-hot encoded labels, and applying identical image preprocessing and label encoding resulted, to an extent, in the image-based self-supervised HCR and HTR models learning relevant information.

RQ2 What is an appropriate loss function for training the image-based self-supervised handwritten text recognition framework?

From the losses considered, the perceptual loss function resulted in the lowest error rates for

both image-based self-supervised HCR and HTR, therefore showing the most potential. However, this image-based loss only has potential for HTR when the input and synthetic images have minimal stylistic differences. In a broader sense, we consider that, for image-based self-supervised HCR, the loss functions directly comparing the input and synthetic image pixels led to substantially higher error rates compared to the perceptual loss, which operates on extracted feature maps. This shows high-level loss functions are necessary for a good performance of image-based self-supervised models.

RQ3. Can pretrained supervised handwritten text recognition models be improved using image-based self-supervised learning?

Image-based self-supervised HTR using a pretrained supervised HTR model showed decreased error rates compared to this pretrained model, indicating that training supervised HTR models can be improved with the proposed framework. However, the results of our experiments demonstrated that this is exclusive to when the stylistic differences between the input and synthetic images are minimal. Otherwise, all information learned by the supervised HTR model will be lost and overwritten, leading to high error rates.

Given the answers to the secondary research questions, we address the primary research question:

Can self-supervised learning based on the similarity between real images and synthetic images that contain predicted text and are produced with handwritten text generation models be effectively used for handwritten text recognition?

The results demonstrated that self-supervised HTR models with both image-based and style-invariant losses for real images did not lead to lower error rates compared to a supervised HTR model. In fact, word error rates were maximal when training the model from scratch. Moreover, when transfer learning was used, catastrophic forgetting occurred. While the style-invariant losses were less affected by catastrophic forgetting, their error rates were still higher compared to the supervised model. However, when trained on synthetic images, image-based self-supervised models learned limited information when trained from scratch, and new information was learned when transfer learning was also used. These findings suggest that the proposed implementation of an image-based self-supervised framework can be effectively used for HTR only if there are minimal stylistic differences between the input and synthetic images, and if background knowledge of HTR is used.

7 Discussion

This Chapter provides a more in-depth discussion of the results and conclusions by addressing the limitations of the methodology and possibilities for future research. To do so, we first consider the HTR and HTG network architectures, and then the implementation and loss functions for the self-supervised framework.

There are several limitations of the self-supervised framework that can be improved upon in future studies on similar systems. Regarding the HTR architecture, the CNN-BLSTM by Puigcerver [16] demonstrated the lowest error rates when adapted with column-wise max pooling [64] and trained with the CE loss instead of the CTC loss. Compared to state-of-the-art word recognition models, however, this was worse. Possible explanations for this are that the supervised HTR model in this thesis was trained with only half the IAM dataset, with different partitions, and without data augmentation. Moreover, it has to be noted that Puigcerver [16]'s original architecture has been shown to be sub-optimal compared to others such as HTR-Flor for line text recognition [18]. Initially, we had considered HTR-Flor's architecture, but due to the incompatibility between its implementation in TensorFlow and GANwriting's implementation in PyTorch, Puigcerver's architecture was opted for instead. Possibly, if ensured to be compatible with the pretrained image generator, HTR-Flor's architecture might result in improved performances of the image-based self-supervised HTR framework. This suggestion will not have much influence, however, if the discrepancy between real images and the synthetic images generated by GANwriting is not addressed.

GANwriting's generator was limited w.r.t. both style and content. Specifically, it struggled to imitate slanted and cursive writing styles, all-capital words, and letters such as 'm' and 'w'. The experiments conducted with self-supervised HTR on real and synthetic images showed that these limitations were detrimental to self-supervised HTR models's performances on real data. As such, one of the key improvements for the proposed image-based self-supervised system is regarding the quality of synthetic images. One way is to enhance GANwriting's generator by including additional constraints to the generator's loss function to control for style, for example, with an extra patch-level or contextual loss term, which improved performance for HiGAN+ [41]. Alternatively, GANwriting could be replaced with a diffusion model such as WordStylist [5]. Although this will substantially increase the computational costs, diffusion models are the current state-of-the-art in HTG. With such a model, the question of whether image-based self-supervised HTR could be effectively applied to real images could be studied more thoroughly. Next to additional constraints or model replacement, the metric for HTG model selection should be appropriate for measuring the quality of handwritten images, as the commonly used FID suffers from biases. Overall, it should be investigated which metrics are fit for HTG so that a consensus can be reached in the field.

Having considered the HTR and HTG architectures, the methods applied for their integration could also be further optimized. First, the image preprocessing might have been sub-optimal for the HTR model. Since inverted images facilitate the image generation process and significantly enhance synthetic image quality, the HTR models were trained with such images. However, in the HTR literature, the models are commonly trained with images that have black ink and a white background. Although the literature does not reason why non-inverted images are used, this trend suggests that HTR architectures can extract more relevant information from them. Therefore, omitting the inversion step in image preprocessing for the HTR pipeline might lead to increased model performances of both the supervised and self-supervised HTR models. Even so, the self-supervised HTR system will likely benefit more from optimizing its objective function.

The image-based self-supervised HTR models trained on synthetic images only showed good performances when leveraging transfer learning. This suggests the image-based losses were too spe-

cific to capture high-level differences in character shapes. Additional constraints, such as projection profiles, might allow the image-based self-supervised models to learn the necessary features. A shortcoming of the perceptual loss, specifically, is that it operates on feature maps extracted with the VGG-16 network pretrained on natural images, and not on handwritten images, likely resulting in sub-optimal feature maps. Perhaps, more relevant features could be extracted by using a pretrained HTR model, or a handwritten word classifier. Such features may result in increased model performances when training without using background knowledge from related tasks as they more explicitly encode textual content. Of course, when extracting feature maps, such losses would still depend on the quality of synthetic images.

Unfortunately, the style-invariant losses (studied on real data) did not result in the learning of any information, not when the self-supervised HTR models were trained from scratch, nor when using transfer learning. For the Siamese Network loss, this could have been due to the extracted features being too implicit w.r.t. content, and the HTR model exploiting this. An alternative to the Siamese Network loss, then, could be a word classifier as suggested for the perceptual loss. Instead of operating on feature maps, feature vectors could be extracted with an extra dense layer. If trained on a variety of writer styles, such features would also be style-invariant. For the low model performances with text-based content loss, we consider that the pretrained supervised HTR model struggled to generalize to unseen words. Many unseen character sequences must have been encountered while training the self-supervised HTR model, especially when training from scratch. This would have led to the comparison of inaccurate predictions. Additionally, in the transfer learning setting, we consider that the discrepancy between real and synthetic images may have propagated to the predicted character probabilities, resulting in increased error rates compared to the pretrained HTR model. A text-based method more robust to out-of-distribution data might be by combining character detection and identification, which was used or word recognition in [24].

Considering the overall effectiveness of the self-supervised system, it could be questioned if it is realistic for practical use due to the high computational costs necessary to train both the system itself and the HTG model. The findings of this thesis suggest that even if the issues with the differences between input and synthetic images are resolved, the self-supervised HTR system might only be effective for improving existing HTR models. In this case, there might be more efficient methods to improve HTR model performances in post-processing. Staying true to the core idea behind image-based self-supervised learning, perhaps synthetic images with HTR predictions could be compared to input images, not for self-supervised learning but for verification. Such an approach would not need additional training of an HTR model or labels; it would allow for automated verification of transcriptions.

Next to the framework's realism, its effectiveness in addressing the need for annotated labels by HTR can also be questioned. While we have proposed several alternatives to further investigate the effectiveness of the proposed self-supervised HTR framework, there remains the fact that only the training of the HTR framework is self-supervised. To train the HTG model, annotated data is still required. In fact, GANwriting needed both text and style labels. Moreover, the background knowledge necessary for good model performances on synthetic images was acquired via a supervised HTR model, which also required annotated data. In light of this, the self-supervised framework only eliminates the need for labels when training the HTR model. It shows the high complexity of the HTR problem and suggests that to obtain high-quality transcriptions of handwritten texts, annotated labels are, to an extent, necessary for high-quality transcriptions.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] J. C. Aradillas Jaramillo, J. J. Murillo-Fuentes, and P. M. Olmos, "Boosting handwriting text recognition in small databases with transfer learning," in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2018, pp. 429–434.

[3] J. Zdenek and H. Nakayama, "Handwritten text generation with character-specific encoding for style imitation," in *Document Analysis and Recognition - ICDAR 2023*, G. A. Fink, R. Jain, K. Kise, and R. Zanibbi, Eds. Cham: Springer Nature Switzerland, 2023, pp. 313–329.

[4] M. Spoto, B. Wolf, A. Fischer, and A. Scius-Bertrand, "Improving handwriting recognition for historical documents using synthetic text lines," in *Intertwining Graphonomics with Human Movements*, C. Carmona-Duarte, M. Diaz, M. A. Ferrer, and A. Morales, Eds. Cham: Springer International Publishing, 2022, pp. 61–75.

[5] K. Nikolaidou, G. Retsinas, V. Christlein, M. Seuret, G. Sfikas, E. B. Smith, H. Mokayed, and M. Liwicki, "WordStylist: styled verbatim handwritten text generation with latent diffusion models," in *International Conference on Document Analysis and Recognition*. Springer, 2023, pp. 384–401.

[6] P. Krishnan, R. Kovvuri, G. Pang, B. Vassilev, and T. Hassner, "TextStyleBrush: Transfer of text aesthetics from a single example," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 7, p. 9122–9134, jul 2023. [Online]. Available: https://doi.org/10.1109/TPAMI.2023.3239736

[7] T. Lin, B. Horne, P. Tiño, and C. Giles, "Learning long-term dependencies is not as difficult with NARX networks," in *Advances in Neural Information Processing Systems*, D. Touretzky, M. Mozer, and M. Hasselmo, Eds., vol. 8. MIT Press, 1995. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1995/file/f197002b9a0853eca5e046d9ca4663d5-Paper.pdf

[8] A. El-Yacoubi, M. Gilloux, R. Sabourin, and C. Suen, "An HMM-based approach for off-line unconstrained handwritten word modeling and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 8, pp. 752–760, 1999.

[9] S. España-Boquera, M. Castro-Bleda, J. Gorbe-Moya, and F. Zamora-Martinez, "Improving offline handwritten text recognition with hybrid HMM/ANN models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 767–779, 2011.

[10] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 369–376. [Online]. Available: https://doi.org/10.1145/1143844.1143891

[11] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[13] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in Neural Information Processing Systems*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., vol. 21. Curran Associates, Inc., 2008. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2008/file/66368270ffd51418ec58bd793f2d9b1b-Paper.pdf

[14] B. Shi, X. Bai, and C. 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*, vol. 39, pp. 2298–2304, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:24139

[15] P. Voigtlaender, P. Doetsch, and H. Ney, "Handwriting recognition with large multidimensional long short-term memory recurrent neural networks," in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016, pp. 228–233.

[16] J. Puigcerver, "Are multidimensional recurrent layers really necessary for handwritten text recognition?" in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, 2017, pp. 67–72.

[17] T. Bluche and R. Messina, "Gated convolutional recurrent neural networks for multilingual handwriting recognition," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, 2017, pp. 646–651.

[18] A. F. de Sousa Neto, B. L. D. Bezerra, A. H. Toselli, and E. B. Lima, "HTR-Flor: A deep learning system for offline handwritten text recognition," in *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, 2020, pp. 54–61.

[19] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://aclanthology.org/D14-1179

[20] J. Almazán, A. Gordo, A. Fornés, and E. Valveny, "Word spotting and recognition with embedded attributes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 12, pp. 2552–2566, 2014.

[21] P. Krishnan, K. Dutta, and C. Jawahar, "Deep feature embedding for accurate recognition and retrieval of handwritten text," in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016, pp. 289–294.

[22] P. Krishnan and C. V. Jawahar, "Matching handwritten document images," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 766–782.

[23] P. krishnan, K. Dutta, and C. V. Jawahar, "HWNET v3: a joint embedding framework for recognition and retrieval of handwritten text," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 26, no. 4, p. 401–417, dec 2023. [Online]. Available: https://doi.org/10.1007/s10032-022-00423-6

[24] R. Mondal, S. Malakar, E. H. Barney Smith, and R. sarkar, "Handwritten english word recognition using a deep learning based object detection architecture," *Multimedia Tools and Applications*, vol. 81, p. 975–1000, jan 2022. [Online]. Available: https://doi.org/10.1007/s11042-021-11425-7

[25] K. Hwang and W. Sung, "Character-level incremental speech recognition with recurrent neural networks," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 5335–5339.

[26] H. Scheidl, S. Fiel, and R. Sablatnig, "Word beam search: A connectionist temporal classification decoding algorithm," in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2018, pp. 253–258.

[27] J. Michael, R. Labahn, T. Gruning, and J. Zollner, "Evaluating sequence-to-sequence models for handwritten text recognition," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2019, pp. 1286–1293. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICDAR.2019.00208

[28] L. Kang, J. I. Toledo, P. Riba, M. Villegas, A. Fornés, and M. Rusiñol, "Convolve, attend and spell: An attention-based sequence-to-sequence model for handwritten word recognition," in *Pattern Recognition*, T. Brox, A. Bruhn, and M. Fritz, Eds. Cham: Springer International Publishing, 2019, pp. 459–472.

[29] J. Poulos and R. Valle, "Character-based handwritten text transcription with attention networks," *Neural Computing and Applications*, vol. 33, p. 0563–10573, aug 2021. [Online]. Available: https://doi.org/10.1007/s00521-021-05813-1

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[31] L. Kang, P. Riba, M. Rusiñol, A. Fornés, and M. Villegas, "Pay attention to what you read: Non-recurrent handwritten text-line recognition," *Pattern Recognition*, vol. 129, p. 108766, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320322002473

[32] M. Li, T. Lv, J. Chen, L. Cui, Y. Lu, D. Florencio, C. Zhang, Z. Li, and F. Wei, "TrOCR: Transformer-based optical character recognition with pre-trained models," 2022.

[33] R. I. Elanwar, "The state of the art in handwriting synthesis," in *2nd International Conference on New Paradigms in Electronics & information Technology (peit'013), Luxor, Egypt*, 2013.

[34] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv*, vol. abs/1308.0850, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:1697424

[35] E. Alonso, B. Moysset, and R. Messina, "Adversarial generation of handwritten text images conditioned on sequences," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, 2019, pp. 481–486.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014.

[37] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 2642–2651. [Online]. Available: https://proceedings.mlr.press/v70/odena17a.html

[38] S. Fogel, H. Averbuch-Elor, S. Cohen, S. Mazor, and R. Litman, "Scrabblegan: Semi-supervised varying length handwritten text generation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 4323–4332.

[39] L. Kang, P. Riba, Y. Wang, M. Rusiñol, A. Fornés, and M. Villegas, "GANwriting: Content-conditioned generation of styled handwritten word images," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 273–289.

[40] J. Gan and W. Wang, "Higan: Handwriting imitation conditioned on arbitrary-length texts and disentangled styles," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, pp. 7484–7492, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16917

[41] J. Gan, W. Wang, J. Leng, and X. Gao, "HiGAN+: Handwriting imitation gan with disentangled representations," *ACM Trans. Graph.*, vol. 42, no. 1, sep 2022. [Online]. Available: https://doi.org/10.1145/3550070

[42] J. Zdenek and H. Nakayama, "JokerGAN: Memory-efficient model for handwritten text generation with text line awareness," in *Proceedings of the 29th ACM International Conference on Multimedia*, ser. MM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 5655–5663. [Online]. Available: https://doi.org/10.1145/3474085.3475713

[43] H. Wang, Y. Wang, and H. Wei, "AFFGANwriting: A handwriting image generation method based on multi-feature fusion," in *Document Analysis and Recognition - ICDAR 2023*, G. A. Fink, R. Jain, K. Kise, and R. Zanibbi, Eds. Cham: Springer Nature Switzerland, 2023, pp. 302–312.

[44] L. Kang, P. Riba, M. Rusiñol, A. Fornés, and M. Villegas, "Content and style aware generation of text-line images for handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, pp. 8846–8860, 2022.

[45] J. Zdenek and H. Nakayama, "Handwritten text generation with character-specific encoding for style imitation," in *Document Analysis and Recognition - ICDAR 2023*, G. A. Fink, R. Jain, K. Kise, and R. Zanibbi, Eds. Cham: Springer Nature Switzerland, 2023, pp. 313–329.

[46] A. Bhunia, S. Khan, H. Cholakkal, R. Anwer, F. Khan, and M. Shah, "Handwriting transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 1066–1074. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.00112

[47] V. Pippi, S. Cascianelli, and R. Cucchiara, "Handwritten text generation from visual archetypes," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2023, pp. 22 458–22 467. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR52729.2023.02151

[48] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2256–2265. [Online]. Available: https://proceedings.mlr.press/v37/sohl-dickstein15.html

[49] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf

[50] Y. Zhu, Z. Li, T. Wang, M. He, and C. Yao, "Conditional text image generation with diffusion models," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 14 235–14 244.

[51] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 10 684–10 695.

[52] A. Mattick, M. Mayr, M. Seuret, A. Maier, and V. Christlein, "SmartPatch: Improving handwritten word imitation with patch discriminators," in *Document Analysis and Recognition – ICDAR 2021*, J. Lladós, D. Lopresti, and S. Uchida, Eds. Cham: Springer International Publishing, 2021, pp. 268–283.

[53] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[54] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 12 1998.

[55] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using DropConnect," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066. [Online]. Available: https://proceedings.mlr.press/v28/wan13.html

[56] K. Cheng, R. Tahir, L. K. Eric, and M. Li, "An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset," *Multimedia Tools and Applications*, vol. 79, no. 19, 2020.

[57] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[58] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[59] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6629–6640.

[60] N. Khare, P. S. Thakur, P. Khanna, and A. Ojha, "Analysis of loss functions for image reconstruction using convolutional autoencoder," in *Computer Vision and Image Processing*, B. Raman, S. Murala, A. Chowdhury, A. Dhall, and P. Goyal, Eds. Cham: Springer International Publishing, 2022, pp. 338–349.

[61] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," 2016.

[62] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[63] U. Marti and H. Bunke, "The IAM-database: an English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, pp. 39–46, nov 2002.

[64] G. Retsinas, G. Sfikas, B. Gatos, and C. Nikou, "Best practices for a handwritten text recognition system," 2024.

[65] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1510–1519.

[66] J. Michael, R. Labahn, T. Grüning, and J. Zöllner, "Evaluating sequence-to-sequence models for handwritten text recognition," *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1286–1293, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:81985132

[67] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 01 2009.

[68] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 47–57, 2017.

[69] B. K. Barakat, R. Alasam, and J. El-Sana, "Word spotting using convolutional siamese network," in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, 2018, pp. 229–234.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[71] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

Appendices

A A Siamese Network for a Style-Invariant Loss

This section details the methodology and results of the Siamese Network used for the Siamese Network loss in the self-supervised HTR framework (Section 4.4). This approach was inspired by Barakat et al. [69], where a Siamese Network was trained for word spotting. They showed that image representations learned by a Siamese Network could be used to rank the similarity of pairs of word images. It does this by mapping pairs of input images to their respective feature representations, using the same network for both inputs. For its application to self-supervised HTR, the aim was thus to train a Siamese Network that can extract word image representations such that real and synthetic images were similar if they contained the same text and dissimilar if they contained different texts, thereby circumventing stylistic differences. We first describe the dataset for training the Siamese Network in Section A.1. Then, the network architecture and its implementation details are given in Sections A.2 and A.3. Lastly, the model evaluation and results are described in Sections A.4 and A.5.

A.1 Dataset

When contemplating the design of the dataset to train the Siamese Network with, it has to be considered that the IAM-HTR dataset was used for training the self-supervised HTR system. To maintain independence between the Siamese Network and this dataset, it was trained on the IAM-GEN dataset. Unlike IAM-HTR, the data splits of IAM-GEN are mutually exclusive w.r.t. writer styles. Therefore, the training and validation sets of IAM-GEN were merged and refactored into a training (70%, 21907 samples), validation (15%, 4565 samples), and test set (15%, 4824) such that each data split contained all writer styles. This dataset is referred to as IAM-GEN-SIA. To create these splits, the same procedure as when creating the IAM-HTR-IV dataset was followed. Moreover, the preprocessing was consistent with the HTR and HTG pipelines to ensure compatibility with these models.

In order for the Siamese Network to learn to extract feature representations of the word images such that a pair of real and synthetic images containing the same text are similar, and dissimilar when containing different texts, we created two similar, and two dissimilar image pairs. These pairs were generated for each real image in the IAM-GEN-SIA dataset. The synthetic images were generated with GANwriting's best-performing pretrained generator. To create the positive (similar, 1) pairs, two images were generated that contained the same text as the real image, s.t. one image contained the same writer style, and one image a different, randomly selected, writer style. To create the negative (dissimilar, 0) pairs, two images were generated in the same writer style as the corresponding real image, but with different texts. Specifically, one image contained a randomly generated text label, and one image contained the text of the real images that was edited. In addition, there was a 25% chance of duplicating characters in both these text labels. Creating the text labels of the negative pairs in this manner simulates the possible situations that could occur during the training of the self-supervised HTR framework. Examples of positive and negative image pairs are shown in Figure 50. The processes of obtaining the text labels for the negative pairs can be detailed as follows.

Random generation For the random generation of the text labels, a vocabulary consisting of the 52 upper and lower case alphabetical characters was used. A random number of $N = \{0, 1, 2, .., 10\}$ characters were then randomly sampled from this vocabulary, weighted by their frequencies in the IAM-GEN-SIA training set.

Label edits One of the synthetic images in each negative pair contained text that was obtained by consecutively applying three random character edits to the text label of the corresponding

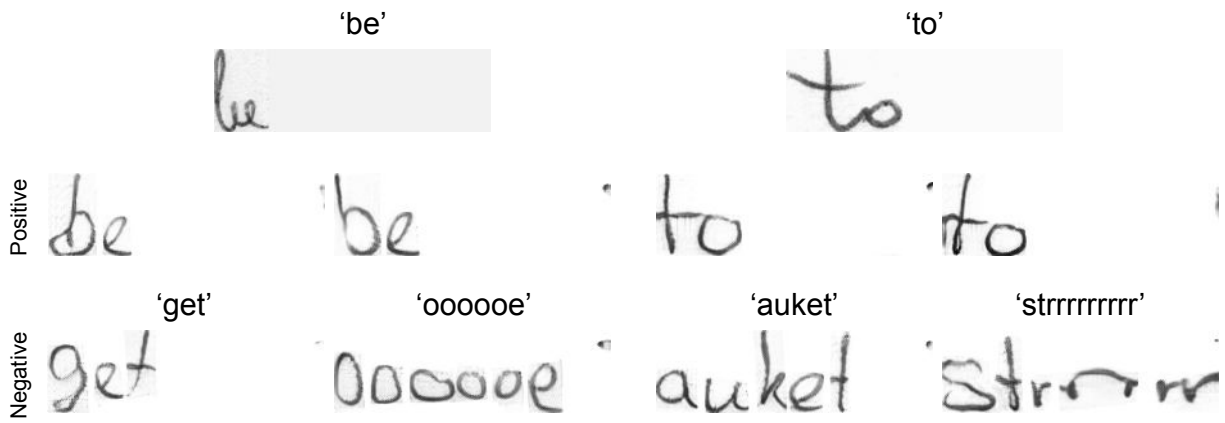


Figure 50: Samples of the synthetic images for the positive and negative pairs with their real counterparts.

real image. These edits were one of random insertion, deletion, swapping, and replacement. For the random insertion and replacement operations, a character was randomly sampled from the vocabulary, weighted by the character frequencies in the IAM-GEN-SIA training set.

Duplication The duplication of characters was applied to the randomly generated and edited labels with a 25% chance. If the duplication occurred, one or two characters (50% chance) in the text label were randomly selected and duplicated a random number of $N = \{2,3,4,5\}$ times.

A.2 Network Architecture

There are two common ways for a Siamese Network to learn a latent space for feature representations: via binary classification (an input pair is the same class or not), or via an explicit distance calculation. The latter method was used in [69], and facilitates interpretation. Hence, we also used this method. An overview of the Siamese Network architecture is shown in Figure 51. It has two branches with the same network architecture, whose weights are shared. Based on initial experimentation, the CNN of the ResNet34 [71] architecture pretrained on ImageNet was selected for the extraction of feature maps. These were then reduced to one-dimensional representations through adaptive average pooling. Then, two dense layers with 256 neurons were used to obtain a final feature representation of the input image. To the first dense layer, the ReLU activation function was applied, as well as batch normalization for regularization.

A.3 Implementation Details

As aforementioned, the weights between the branches of the two input images were shared, meaning that the same network was applied separately to the two input images to extract their feature representations. The network was trained with the contrastive loss function depicted in Equation 19, where $D(x_1,x_2)$ is the Euclidean distance between the feature vectors of input images x_1 and x_2 with the label y, and m = 1.0. The left term of the loss then penalizes large distances between similar pairs, and the right term penalizes small distances between dissimilar image pairs.

$$L_{contr}(\mathbf{x}_1,\mathbf{x}_2,y) = y \cdot D(x_1,x_2)^2 + (1-y)(max(0,m-D(x_1,x_2)))^2 \tag{19}$$

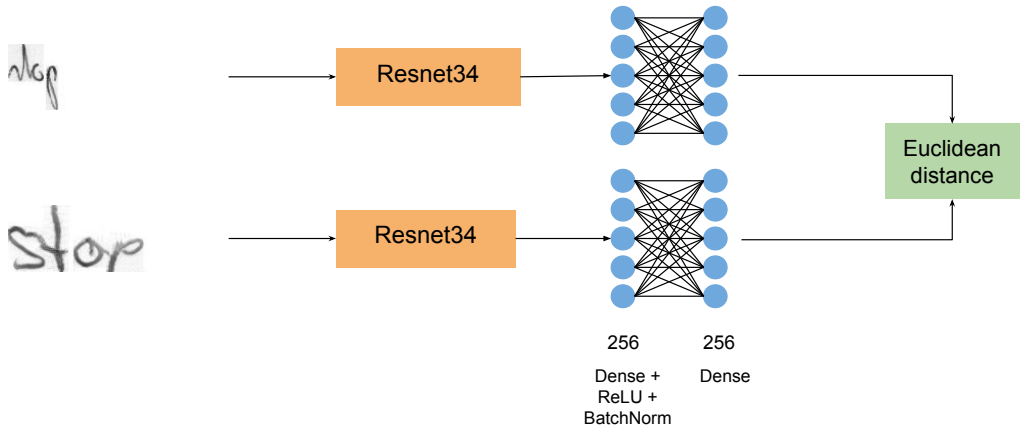


Figure 51: Siamese Network architecture

The Siamese Network was trained with the RMSprop optimizer and an initial learning rate of 0.0001. The learning rate was decreased by a factor of 0.1 when the validation loss did not decrease for more than two epochs. The batch size was set to 16. These parameters were selected through tuning on the validation set.

A.4 Evaluation

The Siamese Network was trained on the IAM-GEN-SIA dataset and evaluated on its validation and test set. Early stopping was applied with a patience of five epochs to prevent overfitting. To evaluate the model's performance, the Euclidean distances between the positive and negative pairs were computed. Specifically, the mean distance across batches on the validation and test set was computed. To verify whether the difference in distance between the positive and negative pairs was statistically significant, a two-tailed t-test was performed on the test set.

A.5 Results

The results of the Siamese Network trained on the IAM-GEN-SIA dataset with the contrastive loss are shown in Table 4 for the validation and test set. On both validation and test sets, there is a clear difference in distance between the similar and dissimilar images. Moreover, the standard deviations indicate only little variation between image pairs, suggesting that appropriate feature representations were learned. Additionally, it can be observed that there is a minimal difference between the results on the validation and test sets, suggesting that the model is able to generalize well to unseen data. Considering the loss curve in Figure 52, it can be observed that only minimal overfitting occurred. A two-tailed t-test conducted on the test results to compare the mean distances of similar and dissimilar images indicated that the distance between similar images was significantly smaller than the distance between dissimilar images ($t(2282) = 161.024$, $p < 0.005$).

Table 17: Siamese network results on the IAM-GEN-SIA dataset.

	Euclidean distance Validation set	Euclidean distance test set
Similar images	0.105 ± 0.069	0.107 ± 0.070
Dissimilar images	0.541 ± 0.054	0.539 ± 0.057
Difference	0.435 ± 0.085	0.432 ± 0.089

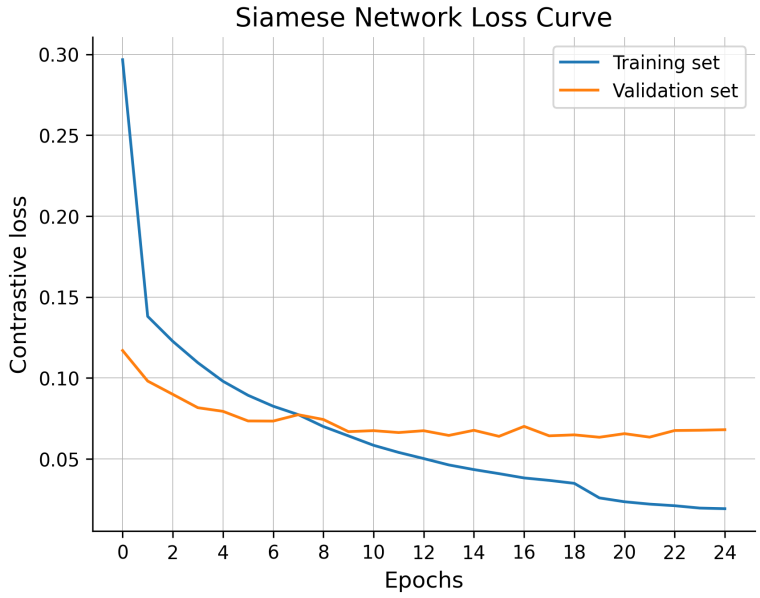


Figure 52: Training and validation loss curves of the Siamese Network on IAM-GEN-SIA.

B Results of Preliminary Self-Supervised HTR Experiments with Different Learning Rates on Real IAM-HTR data.

This section reports and briefly discusses the results of experiments with self-supervised HTR trained on real IAM-HTR data in the pretrained setting with learning rates of 1×10^{-4} and 3×10^{-4}. The results are shown in Table 18. The model performances for the SSIM and Perceptual losses showed a slight decrease in CER and WER. However, this is likely due to randomness in the image generation process during training, rather than a structural improvement due to an increased learning rate. We consider this explanation, as the style-invariant losses showed a drastic decrease in model performance with this increase in learning rate from 1×10^{-4} to 3×10^{-4}.

Given the above, it was concluded that, for real images, the initial learning rate of 1×10^{-4} led to improved self-supervised HTR model performances compared to the initial learning rate of 3×10^{-4}.

Table 18: Self-supervised HTR model performances with different learning rates on real IAM-HTR validation data. The models used transfer learning and were trained on real IAM-HTR data.

		IV-Validation		**OOV-Validation**	
Loss Function	**Learning Rate**	CER (%)	WER (%)	CER (%)	WER (%)
SSIM	1×10^{-4}	95.64	100	95.06	100
	3×10^{-4}	95.48	100	94.85	100
Perceptual	1×10^{-4}	94.49	100	94.22	100
	3×10^{-4}	94.22	100	93.70	100
Siamese Network	1×10^{-4}	59.89	81.98	74.94	99.89
	3×10^{-4}	98.60	100	97.89	100
Content	1×10^{-4}	21.26	48.50	39.66	88.86
	3×10^{-4}	95.13	100	95.10	100

C Results of Preliminary Self-Supervised HTR Experiments with Different Learning Rates on Recreated IAM-HTR data.

This section reports and briefly discusses the results of experiments with image-based self-supervised HTR trained on recreated IAM-HTR data in the pretrained setting with learning rates of 1×10^{-4} and 3×10^{-4}. The results are shown in Table 19. Comparing the results for the different learning rates, the model performances for both the SSIM and Perceptual losses with learning rate 3×10^{-4} showed a decrease in CER and WER compared to the learning rate of 1×10^{-4}. The more substantial decreases in WER on the OOV validation data for both losses lead us to believe that this can be attributed to the increased learning rate rather than the randomness in the image generation process.

Given the above, it was concluded that, for synthetic images, the initial learning rate of 3×10^{-4} led to improved image-based self-supervised model performances compared to the initial learning rate of 1×10^{-4}.

Table 19: Image-based self-supervised HTR model performances with different learning rates on synthetic IAM-HTR validation data. The models used transfer learning and were trained on recreated IAM-HTR-IV data.

		IV Validation		**OOV Validation**	
Loss function	**Learning Rate**	CER (%)	WER (%)	CER (%)	WER (%)
SSIM	1×10^{-4}	0.94	3.64	5.07	22.16
	3×10^{-4}	0.70	2.71	2.28	9.32
Perceptual	1×10^{-4}	0.44	1.84	1.90	8.75
	3×10^{-4}	0.36	1.45	1.46	6.48

D Results of Image-based Self-Supervised HTR for Learning New Words

This section reports the results w.r.t. error rates of the experiments conducted with image-based self-supervised HTR on the recreated IAM-HTR dataset in order to explicitly investigate whether the proposed framework is able to learn new words. Specifically, the image-based self-supervised models were trained on the IAM-HTR-OOV dataset with the SSIM and perceptual loss functions and evaluated with both IV and OOV validation data. Additionally, we test the performance of the best-performing model on the real IAM-HTR data in comparison to the best-performing supervised HTR model (trained on real IAM-HTR data). As the results of the self-supervised models trained on IAM-HTR-IV indicated, transfer learning was necessary for the learning of new features with the image-based loss functions (see Section 5.3), and this was also used in the current experiments.

Table 20 shows the results of the image-based self-supervised HTR models trained on IAM-HTR-OOV data for the IAM-HTR validation sets. Overall, the results show decreased error rates for both SSIM and perceptual loss functions compared to the pretrained supervised HTR model, indicating new relevant features were learned for both IV and OOV data. Additionally, the SSIM loss function led to decreased error rates on the IV validation set compared to the perceptual loss function, while this was reversed for the OOV validation set. Since the difference in performance on the IAM-HTR-IV validation set is much smaller compared to the differences on the IAM-HTR-OOV validation set, and these experiments focus on the latter, the image-based self-supervised model trained with the perceptual loss was determined to be the best-performing model in the current experimental setup.

The error rates of the best-performing image-based self-supervised model trained on recreated IAM-HTR-OOV data in comparison to the best-performing supervised HTR model trained on real IAM-HTR-IV are shown in Table 21. First, we can observe a substantial increase in error rates of the self-supervised model's performance on real test data compared to recreated test data, which is in line with the findings of Section 5.3. Second, it can be observed that training on recreated OOV data led to increased model performance on real IAM-HTR-OOV test data compared to the supervised HTR model. However, this was reversed for the real IAM-HTR-IV test set. Still, this may suggest that new information was learned with the recreated IAM-HTR-OOV data that was also relevant to real data. Another possibility is that the IAM-GEN-SIA contained some of the words that are in the OOV dataset, which was learned by the pretrained supervised model and maintained by the self-supervised model.

Table 20: Image-based self-supervised model performances on synthetic validation data. Both self-supervised models used transfer learning and were trained on recreated IAM-HTR-OOV data. The pretrained model was trained on real images of IAM-GEN-SIA.

	IV Validation		**OOV Validation**	
Loss function	CER (%)	WER (%)	CER (%)	WER (%)
SSIM	**3.48**	**12.93**	2.57	8.86
Perceptual	3.81	13.66	**1.29**	**5.68**
CE (pretrained HTR)	5.60	17.05	14.84	49.65

Table 21: Image-based self-supervised model performance for the perceptual loss on the IAM-HTR test data in comparison with the best-performing supervised HTR model trained on real IAM-HTR-IV data. The self-supervised model was trained on recreated IAM-HTR-OOV data.

			IV Test		**OOV Test**	
Training Data	**Test Data**	**Loss Function**	CER (%)	WER (%)	CER (%)	WER (%)
OOV	Real IAM-HTR	Perceptual	34.33	64.31	36.63	79.46
IV		CE	14.71	30.89	44.67	93.13
OOV	Recreated IAM-HTR	Perceptual	3.84	13.15	1.36	5.41