

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**SINGLE SAMPLE STYLE TRANSFER LEARNING FOR
SYNTHESIZING HANDWRITING**

ÖZDENUR UÇAR
A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

GEBZE
2022

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**SINGLE SAMPLE STYLE TRANSFER
LEARNING FOR SYNTHESIZING
HANDWRITING**

ÖZDENUR UÇAR
**A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING**

THESIS SUPERVISOR
ASSIST. PROF. DR. YAKUP GENÇ

GEBZE
2022

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

EL YAZISI ÜRETİMİ İÇİN TEK
ÖRNEKLE STİL TRANSFER ÖĞRENİMİ

ÖZDENUR UÇAR
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
DR. ÖĞR. ÜYESİ YAKUP GENÇ

GEBZE
2022

GEBZE TEKNİK ÜNİVERSİTESİ	YÜKSEK LİSANS JÜRİ ONAY FORMU
----------------------------------	--------------------------------------

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 07/07/2022 tarih ve 2022/34 sayılı kararıyla oluşturulan jüri tarafından 27/07/2022 tarihinde tez savunma sınavı yapılan Özdenur Uçar Çakıroğlu'nun tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE

(TEZ DANIŞMANI) : Dr. Öğr. Üyesi Yakup GENÇ

ÜYE

: Doç. Dr. Mehmet GÖKTÜRK

ÜYE

: Dr. Öğr. Üyesi Tarkan AYDIN

ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun
...../...../..... tarih ve/..... sayılı kararı.

İMZA/MÜHÜR

SUMMARY

Handwritten character generation is a popular topic with a variety of applications. Many methods of character generation have been proposed in the literature, but few of these methods focus on preserving the writer's style while generating handwritten characters. Representing handwriting styles involves the challenge of representing both the style of each character and the overall style of the author. In this study, unlike the studies in the literature, it is tried to produce the characters of the person that we have not seen yet by extracting the handwriting style of the person with limited data. While generating handwritten characters, spline curves of the characters were used as input, as well as handwritten images. In the developed method, a multitasking learning network is proposed by using the Conditional Variational Autoencoder (CVAE) model, which is one of the deep learning methods. In the experiments carried out, three different models were compared, and it was seen that the results of the Conditional Variational Autoencoder (CVAE) network trained with spline curves were better than the other models.

Key Words: Handwritting Generation, Conditional Variational Autoencoder, Writing Style, Spline Curve.

ÖZET

El yazısı karakter oluşturma, çeşitli uygulamalarla popüler bir konudur. Literatürde birçok karakter oluşturma yöntemi önerilmiştir, ancak bu yöntemlerin çok azı el yazısı karakterler üretirken yazarın stilini korumaya odaklanır. El yazısı stillerini temsil etmek, hem her bir karakterin stilini hem de yazarın genel stilini temsil etme zorluğunu içerir. Bu çalışmada literatürdeki çalışmalardan farklı olarak sınırlı veri ile kişinin el yazısı üslubu çıkarılarak henüz görmediğimiz kişinin karakterleri üretilmeye çalışılmıştır. El yazısı karakterler oluşturulurken, girdi olarak karakterlerin spline eğrileri ve el yazısı görüntüleri kullanılmıştır. Geliştirilen yöntemde derin öğrenme yöntemlerinden Koşullu Değişken Otomatik Kodlayıcı (CVAE) modeli kullanılarak çok görevli bir öğrenme ağı önerilmiştir. Yapılan deneylerde üç farklı model karşılaştırılmış ve spline eğrileri ile eğitilmiş Koşullu Değişken Otomatik Kodlayıcı (CVAE) ağının sonuçlarının diğer modellere göre daha iyi olduğu görülmüştür.

Anahtar Kelimeler: El Yazısı Karakter Üretimi, Koşullu Varyasyonel Otokodlayıcı, Yazım Stili, Spline Eğrileri.

ACKNOWLEDGEMENTS

I am grateful to my advisor, Assist. Prof. Yakup Genç, for not only sharing his scientific knowledge but also encouraging me throughout our study. Thanks to his patient sharing of his extensive knowledge and experience, I gained a better research perspective.

My special thanks are for my mom, Mevlüde Uçar, who always supported me and believed that I could manage to obtain any achievement I desire for. This work could not have been completed without her support.

I am grateful to my dear friends Selen Postacı and Farukcan Şoray, who always supported me throughout the work and believed that I will complete this work with their supportive attitude, it is a great chance for me to have friends like them.

Last but not least, I am deeply grateful to the love of my life, my husband, Alperen Çakıroğlu, for always being supportive and guiding throughout the study. I would not have been able to complete this work without his support. Thank you so much for making my life beautiful in every way and always making me happy.

TABLE of CONTENTS

	<u>Page</u>
SUMMARY	iv
ÖZET	v
ACKNOWLEDGMENTS	vi
TABLE of CONTENTS	vii
LIST of ABBREVIATIONS and ACRONYMS	viii
LIST of FIGURES	ix
LIST of TABLES	xi
1. INTRODUCTION	1
2. CHARACTER SYNTHESIS	4
3. STYLES, SPLINES AND CHARACTER GENERATION	7
3.1. Computation of Spline Curves	7
3.2. Autoencoders	8
3.2.1. Variational Autoencoder	11
3.2.2. Conditional Variational Autoencoder	15
3.3. Proposed Architecture	17
4. EXPERIMENTS AND RESULTS	24
4.1. Dataset	24
4.1.1. Preprocessing	26
4.1.2. Data Classification	26
4.2. Single-Tasking Architecture	28
4.3. Multi-Tasking Architecture	31
5. CONCLUSION	35
REFERENCES	37
BIOGRAPHY	41
APPENDICES	42

LIST of ABBREVIATIONS and ACRONYMS

<u>Abbreviations</u>	<u>Explanations</u>
<u>and Acronyms</u>	
2D	: 2 Dimensional
CNN	: Convolutional Neural Network
CVAE	: Conditional Variational Autoencoder
DL	: Deep Learning
GAN	: Generative Adversarial Network
KL	: Kullback-Leibler
LSTM	: Long Short-Term Memory
MSE	: Mean Squared Error
ReLU	: Rectifier Linear Unit
RNN	: Recurrent Neural Network
VAE	: Variational Autoencoder

LIST of FIGURES

<u>Figure No:</u>	<u>Page</u>
2.1: Generating handwritten letters in different styles.	5
3.1: From left to right; the spline curve created with the raw handwritten character, the skeleton image of the character, and the control points.	8
3.2: Design of basic auto encoder.	10
3.3: The basic scheme of a variational autoencoder.	13
3.4: Scheme of a variational autoencoder after a reparameterization trick.	14
3.5: Graphical model of CVAE network.	16
3.6: The basic scheme of a conditional variational autoencoder.	17
3.7: Architecture of the proposed conditional variational autoencoder network.	18
3.8: Deconvolution based decoder.	21
3.9: An illusion of the deconvolution process.	21
3.10: The proposed network including spline curves.	22
4.1: Example 5 and 6 characters from the NIST19 dataset by the same writer.	25
4.2: Samples of the same author from the A and B sets created for the training set in the NIST 19 dataset.	25
4.3: From left to right, a character in the NIST19 dataset and a character formed after preprocessing.	26
4.4: The architecture of the network used for character classification in the NIST19 dataset.	28
4.5: Loss values during training and validation.	30
4.6: Generated CVAE model with different inputs from one author the outputs that happened.	32

LIST of TABLES

<u>Table No:</u>	<u>Page</u>
3.1: The properties of the convolution layers of the encoder network in the proposed method.	20
3.2: The properties of the convolution layers of the decoder network in the proposed method.	22
4.1: The value range of hyperparameters.	28
4.2: Initial parameters for CNN based CVAE model.	29
4.3: Fine-tuned training parameters and values	29
4.4 Test results of the CVAE model.	30
4.5: Initial settings of the spline based CVAE network.	32
4.6: PSNR and SSIM of handwritten characters but the others.	34

1. INTRODUCTION

One of humanity's greatest achievements is the documentation of knowledge through handwriting: the oldest written documents represent the shift from prehistory to history, and most proof of historic events can be found in handwritten scripts and markings. After Gutenberg's printing press in the mid-1400s, handwriting remained the dominant method of recording events and data. In the digital era, when courtroom stenographers are replaced by technology [5], both printing and handwriting are becoming somewhat obsolete. Furthermore, much of the content we type remains in digital form and never encounters paper. Nonetheless, handwritten text has a wide range of uses today; a large volume of handwritten text has gathered through time, ready to be processed, and is continually being written today. Healthcare and banking industries are two famous examples of where handwriting is still employed today. There is an increasing demand for these to be retrieved and made available, for example, through modern search engines. Modern object character recognition (OCR) appears to be capable of handling printed text, however handwritten text recognition (HTR) appears to be lacking.

With the development of deep learning, handwritten text recognition has reached an unprecedented stage of development; however, many training examples are needed to create a recognition dictionary. By increasing the number of characters used, more variations of handwritten characters are used, improving recognition performance. When there are not enough character samples in the existing handwritten datasets, data augmentation is provided by character generation. There are two separate ways to present a character. One approach is to think of it as a collection of pixels, while another is to think of it as a series or combination of strokes. The former is also known as a character's image. In terms of the latter, we can see the basic elements (strokes) that a character has and how they are connected in the sequence. Furthermore, we can use these sequences to construct new characters. However, generating character in this way will encounter many problems. One significant problem is that we'll probably have a lot of elements, some of which are only appropriate for certain characters, or we might end up with unnatural characters. Another issue is that the similarity between origin and generated characters is difficult to determine. As a result, considering the characters as images is a more practical option for character creation. Convolutional

neural net (CNN) is the most widely used machine learning method for image-related tasks such as object recognition and image classification. CNN learns to reduce the size of a loss function that is used to assess the quality of output. The CNN model succeeds at this job because it typically has more configurable parameters than other models, which means it can suit the goal function, which can be quite complex. We must tell a CNN model which objective must be minimized to train it; consequently, this model is supervised learning. According to the literature, the CNN model can manage not only discriminate problems such as image classification, but also generation challenges.

For a general character recognition problem, character generation gives very good results, but enough datasets to recognize personal characters and the number of labeled data to represent the person's style are very few. In many cases, samples of personal handwritten characters are difficult to obtain, such as in historical documents. To avoid this problem, generating author-based handwriting samples has become an important research topic. One of the most effective ways to obtain character samples that carry a personal style is to generate character data through style transfer. In the past years, many different studies have been carried out on the generation of handwritten characters. Generative Adversarial Networks (GAN) and Variational Autoencoder Networks (VAE) have made great progress in this regard. A few years after the emergence of the concept of Generative Adversarial Network [1], impressive progress has been made in generating realistic artificial images [5]. But in the original GAN architecture, the inputs are randomly sampled from the latent space, so it is difficult to control what kind of images are generated. For this reason, the generative process of the GAN and VAE models is created to allow conditioning on a class label and several types of images are generated from the image given as input. Among these deeply generative models, Conditional Variational Autoencoder (CVAE) [6] and Conditional Generative Adversarial Network (CGAN) [9] are the most typical and popular, leading, respectively.

Our purpose in this study is to generate characters that are not in the training set from a given limited number of characters by using the Conditionally Variable Autoencoder (CVAE) network, which is one of the generative models. There are two types of datasets, online and offline, for generating handwritten characters. Online datasets contain handwritten (x, y) coordinates and strokes. Offline datasets consist of handwritten images. In our problem, we use the NIST19 dataset, which is one of the

offline datasets. The NIST19 dataset contains 128x128 numbers and alphabetic characters from full-page binary images of 3699 Handwriting Sample Forms (HSF). When creating NIST19, it was composed to include only class (character, number, uppercase and lowercase) and writer information. There are multiple instances of each character from a writer in the dataset. Due to the nature of handwriting, the characters in the figure are not the same, even if they belong to the same person. For this reason, we use each of the sample characters given while extracting the style. Since character labels are used together with writer labels in our problem, character classification is required and using a vggnet-based model, the character classification process has been done with an accuracy value of 99.2%.



2. CHARACTER SYNTHESIS

In human society, handwriting is very essential. As a result, handwritten character generation is a popular artificial intelligence research area. Researchers offered various methods for generating handwritten characters. Machine learning-based methods have recently gained widespread acceptance in industry and academia, owing to the rapid growth of hardware and its generalization capacity. Traditional handwriting generation methods typically include isolated characters; involves costly manual operations requiring strong domain-specific knowledge aimed at clipping, creating backgrounds and ligatures between strokes. With the remarkable success of deep learning in machine learning, artificial neural networks gradually entered the problem of handwriting synthesis. The first related work is that Graves [1] proposed generating online handwritten trajectories with Recurrent Neural Networks (RNNs). Alex's Generator encodes a set of parameters for a probability distribution of the next stroke point at each timestamp, then samples the next stroke point based on this distribution. Alex's Generator comes in two flavors: handwritten predictor and handwritten synthesizer, the latter of which can synthesis handwritten text for a given text.

Inspired by the work of Graves, Aksan et al. [8] proposes a method based on conditional Variable RNN(VRNN), in which the input is split into two separate hidden variables to represent content and style. Their method is capable of synthesizing handwritten text from a series of digital ink or ASCII characters. The suggested model is capable of learning and reproducing local variations in handwriting, as well as precisely imitating the writing styles of various users. They present variational autoencoder [7] architecture as well as two latent distributions that enable conditional synthesis, a unique training and sampling technique. The proposed system was trained on handwritten text samples segmented by 294 authors from the IAM-OnDB dataset. As illustrated in Figure 2.1, the system can transmit styles across users and develop innovative sequences from entered text.

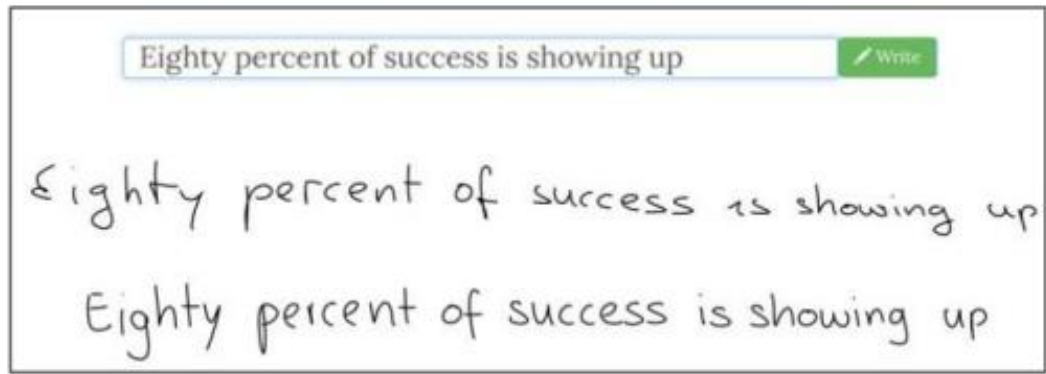


Figure 2.1: Generating handwritten letters in different styles.

However, their approach tends to average certain styles across authors, thus reducing detail [11]. In a later work, the VRNN module replaced with Stochastic Temporal CNNs, which have been shown to provide more consistent handwriting production. According to Kotani et al. [8] proposes an online handwritten stroke representation approach to represent latent style information by encoding author, character, and author-character-specific style changes in an RNN mode.

In such iteration-based generative models, besides the difficulty of learning long-range dependencies, it is also time-consuming due to the computational mechanism. Generative sequential models use handwritten stroke data, obtaining these stroke data is practically more difficult than processing the handwritten image [8]. In contrast, it is much easier to get a handwritten image in real life and on the internet. Due to recent breakthroughs in deep learning approaches, generating handwritten characters/text from handwriting images has become increasingly popular. Following the introduction and early application of GANs to the generation of Latin handwritten numerals [8, 9], recent studies have concentrated on the generation of handwritten characters and words. It has been shown that both GANs [20] and variational autoencoders (VAE) [17] can generate images of realistic handwritten digits. Later, Mirza et al. proposed conditional GANs (cGANs) [6] that can generate images of single handwritten digits conditioned in their class labels.

Haines et al. [12] presents an approach for generating new text in a distinctive style extracted from source images. Their models require a certain degree of human intervention during character segmentation and are limited to generating characters found in source images. The work of [10] uses CycleGAN [10] to synthesize images of

isolated handwritten characters of the Chinese language. Alonso et al. [11] Suggesting an approach where handwritten text rendering is conditioned by strings. However, their approach suffers from style collapse that hinders the diversity of synthesized images. Fogel et al. [2] proposes an approach called ScrabbleGAN, which synthesizes handwritten words using a fully convolutional architecture. Characters generated here have similar recipient field widths. [7] introduced a conversion model using online handwriting from offline samples, and then the technique of transferring style to online data. This approach relies on the performance of the transform model. More recently, Kang et al. [13] proposed GANwriting, which can generate short handwritten words based on the style images provided. However, their methods are limited to synthesizing short handwritten words (less than ten letters) and therefore cannot generate long handwritten texts.

3. STYLES, SPLINES AND CHARACTER GENERATION

In this work, we designed a CNN-based Conditional Variational Autoencoder (CVAE) network for the style-based character generation problem. In the network we have designed, we discovered that spline curves also contribute to learning the style while extracting the features of a character. While the "encoder" network in the autoencoder is being trained, two networks that both learn the spline curves and feature extraction of the handwritten character given as input are trained simultaneously with the multi-tasking learning (MTL) method. In the sub-steps, the structure of the CVAE network and the creation of parametric b-spline curves are explained. In this section, the details about the dataset, data preprocessing methods and the learning models were discussed.

3.1. Computation of Spline Curves

All characters are generated by curves during the writing or signing process; each word or letter can be considered a curve. B-spline is one of the most efficient curve representations in geometric modeling. B-Spline curves are useful for simulating handwriting, printed texts, or signatures of people with varied spellings since they are stable under affine transformations such as translation, scaling, and rotations. A set of control points does not adequately characterize the B-spline curve. The challenge of recognizing the matching set of control points of each character arises from the building of a B-Spline curve for each character. We create the segmented spline curves of each character for the style inference of the person involved in our problem. Creating spline curves for characters follows the steps below:

- Conversion of character images to binary images.
- Binary the character bitmaps or transform the image of characters into a bit matrix (convention that black pixel is 0 and white pixels is 1).
- Extracting the skeleton of the character from binary images. The goal of skeletonization is to determine a character's medial axis. Skeletonization is the technique of removing as many pixels from a pattern as feasible without

changing its overall shape. The design should still be recognizable after pixels have been smoothed.

- Reducing the number of points representing the character skeleton with the Peucker [5] algorithm.
- Calculation of the piece-wise b-spline curve from the obtained coordinates.

In Figure 3.1, the spline curve obtained on an example handwriting character is shown.



Figure 3.1: From left to right; the spline curve created with the raw handwritten character, the skeleton image of the character, and the control points.

3.2. Autoencoders

A sort of artificial neural network called an autoencoder is used to learn efficient coding for unlabeled input (unsupervised learning). By attempting to regenerate the input from the encoding, the encoding is validated and enhanced. By training the network to disregard inconsequential data ("noise"), the autoencoder learns a representation (encoding) for a set of data, generally for dimensionality reduction. It tries to reconstruct the original input by learning how to compress and reconstruct the dataset in the most efficient way. In a standard autoencoder, the dimensions of the inputs and outputs are equal. During the training, it is also referred to as a self-supervised architecture as it generates its own tags. It was suggested in an article published in 1986 [18]. So far, research areas have expanded, and many applications have been made on this architecture. As a result, it has turned into a structure where the weights of the outputs can be calculated.

An autoencoder consists of two parts: an encoder that converts the input into code and a decoder that converts the code back into the input. The simplest way to perform the copying task perfectly would be to duplicate the signal. Instead, autoencoders are frequently compelled to approximate the input, maintaining just the most important aspects of the data in the copy. The simplest form of an autoencoder is a feed-forward, non-recurrent neural network that uses an input layer and an output layer interconnected by one or more hidden layers, like single-layer perceptron participate into multi-layer perceptrons (MLP). The output layer has the same number of nodes (neurons) as the input layer. Its purpose is to reconstruct its inputs (minimizing the difference between input and output) rather than predicting a target Y , value given the inputs X . Therefore, autoencoders learn unsupervised.

An autoencoder consists of two parts, the encoder and the decoder which can be defined as transitions ϕ and ψ such that:

$$\begin{aligned}\phi: X &\rightarrow F \\ \psi: F &\rightarrow X \\ \phi, \psi &= \arg \min_{\phi, \psi} |X - (\phi \circ \psi)|^2\end{aligned}\tag{3.1}$$

In the simplest case, given one hidden layer, the encoder stage of an autoencoder takes the input $x \in R^d = X$ and maps it to $h \in R^p = F$.

$$h = \sigma(Wx + b)\tag{3.2}$$

This image $\mathbf{h}$ is usually referred to as code, latent variables, or a latent representation. σ is an element-wise activation function such as a sigmoid function or a rectified linear unit. $\mathbf{W}$ is a weight matrix and $\mathbf{b}$ is a bias vector. The basic autoencoder scheme is shown in Figure 3.2. Weights and biases are usually initialized randomly, and then updated iteratively during training through backpropagation. After that, the decoder stage of the autoencoder maps $\mathbf{h}$ to the reconstruction $\mathbf{x}'$ of the same shape as $\mathbf{x}$:

$$x' = \sigma'(W'h + b')\tag{3.3}$$

Where σ , $\mathbf{W}$ and $\mathbf{b}$ for the decoder may be unrelated to the corresponding σ , $\mathbf{W}$ and $\mathbf{b}$ for the encoder.

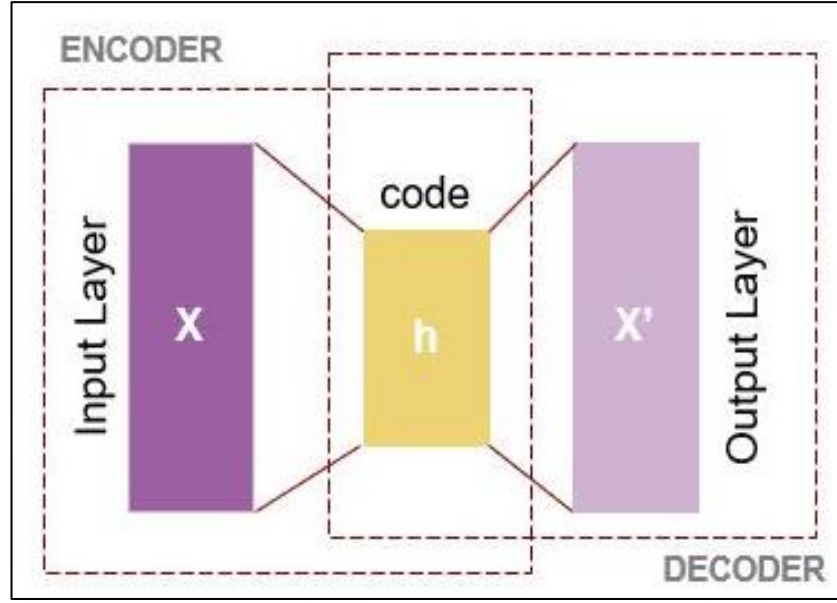


Figure 3.2: Design of basic autoencoder.

Autoencoders are designed to reduce reconstruction errors (such as squared errors), also known as the loss function:

$$L(x, x') = |x - x'|^2 = |x - \sigma'(W'(\sigma(Wx + b)) + b')|^2 \quad (3.4)$$

where $\mathbf{x}$ is usually averaged over the training set.

Autoencoder training, like other feedforward neural networks, is performed by backpropagating the error. The feature space $\mathbf{F}$ must have a lower dimensionality than the input space $\mathbf{X}$. the feature vector $\phi(\mathbf{x})$ can be regarded as a compressed representation of the input $\mathbf{x}$. Undercomplete autoencoders are an example of this. An autoencoder can learn the identity function and become worthless if the hidden layers are greater than (overcomplete) or equal to the input layer, or if the hidden units are given enough capacity. Experiments have shown, however, that overcomplete autoencoders can nevertheless learn useful features [20]. The code dimension and model capacity should, in an ideal world, be determined by the complexity of the data distribution to be modeled. Regularized Autoencoders are a model version that can be used to do this [21].

Learning representations in a way that promotes sparsity increases classification performance. [23] Sparse autoencoders can have more hidden units (rather than fewer) than inputs, but only a limited number of hidden units can be active at the same time (thus, sparse). [24] This constraint pushes the model to adapt to the training data's

unique statistical features. Specifically, a sparse autoencoder is an autoencoder whose training criterion involves a sparsity penalty $\Omega(\mathbf{h})$ on the code layer $\mathbf{h}$.

$$L(x, x') + \Omega(h) \quad (3.5)$$

Recalling that $\mathbf{h} = \mathbf{f}(\mathbf{W}\mathbf{x} + \mathbf{b})$ the basis of the input data, the penalty pushes the model to activate (i.e., output value close to 1) parts of the network, while inactivating all other neurons (i.e., to have an output value close to 0) [20].

3.2.1. Variational Autoencoder

The variational Bayesian (VB) process includes optimizing an approximation to an intractable posterior distribution. VAEs (variational autoencoders) are a type of variational Bayesian approach. Despite their architectural resemblance to simple autoencoders, VAEs are distinct systems with distinct goals and mathematical formulations. Instead of a fixed vector, the hidden space is made up of a mix of distributions. VAE is a deep generative model in which both the decoder and the encoder can be learnt from data concurrently [15, 17]. VAE has the advantage of providing an amortized inference technique for computing a latent representation via the encoder and estimating an implicit density model for a given dataset via the decoder. The decoder is the focus of attention when learning a generative model for data. The encoder, on the other hand, is more important when the goal is to extract meaningful features from the data and learn a good representation [28]. VAE models are often associated with the autoencoder [29] model because of their architectural similarity, but there are significant differences in purpose and mathematical formulation. Statistical inference issues (such as subtracting the value of one random variable from the value of another random variable) can be reformulated as statistical optimization problems using variational autoencoders (i.e., finding parameter values that minimize some objective function). It's used to convert a single variable into a multivariate latent distribution. Although this technique was created for unsupervised learning, its efficacy has been demonstrated for semi-supervised learning [19] and supervised learning [20].

The input data in a VAE is sampled from a parametrized distribution (the prior in Bayesian inference terms), and the encoder and decoder are trained together so that

the output minimizes the Kullback–Leibler [21] divergence between the parametric posterior and the true posterior.

From a formal viewpoint, given an input dataset $\mathbf{x}$ with an unknown probability distribution $\mathbf{P}(\mathbf{x})$, the goal is to model or approximate the true distribution $\mathbf{P}$ of the data using a parametrized distribution $\mathbf{p}(\boldsymbol{\phi})$ with parameters $\boldsymbol{\phi}$. Let $\mathbf{z}$ be a random vector that is distributed in the same way as $\mathbf{x}$. In terms of concept, $\mathbf{z}$ will be a latent encoding of $\mathbf{x}$. Over $\mathbf{z}$, marginalizing gives

$$p_{\theta}(x) = \int_{\mathbf{z}} p_{\theta}(x, \mathbf{z}) d\mathbf{z} \quad (3.6)$$

where represents the joint distribution under $\mathbf{p}_{\theta}(\mathbf{x}, \mathbf{z})$ of the observable data $\mathbf{x}$ and its latent representation or encoding $\mathbf{z}$. According to the chain rule, the equation can be rewritten as

$$p_{\theta}(x) = \int_{\mathbf{z}} p_{\theta}(x|\mathbf{z}) p_{\theta}(\mathbf{z}) d\mathbf{z} \quad (3.7)$$

The vanilla variational autoencoder assumes that $\mathbf{z}$ is a finite-dimensional vector of real numbers, and that $\mathbf{p}_{\theta}(\mathbf{x}, \mathbf{z})$ is a Gaussian distribution. Then $\mathbf{p}_{\theta}(\mathbf{x})$ is a Gaussian distribution mixture.

The collection of relationships between the input data and its latent representation can now be defined as

- Prior $\mathbf{p}_{\theta}(\mathbf{z})$
- Likelihood $\mathbf{p}_{\theta}(\mathbf{x}|\mathbf{z})$
- Posterior $\mathbf{p}_{\theta}(\mathbf{z}|\mathbf{x})$

Unfortunately, computing $\mathbf{p}_{\theta}(\mathbf{x})$ is time-consuming and in most circumstances impossible. To speed up the analysis to make it possible, it is necessary to add another function that approximates the posterior distribution as follows

$$q_{\Phi}(z|x) \approx p_{\theta}(z|x) \quad (3.8)$$

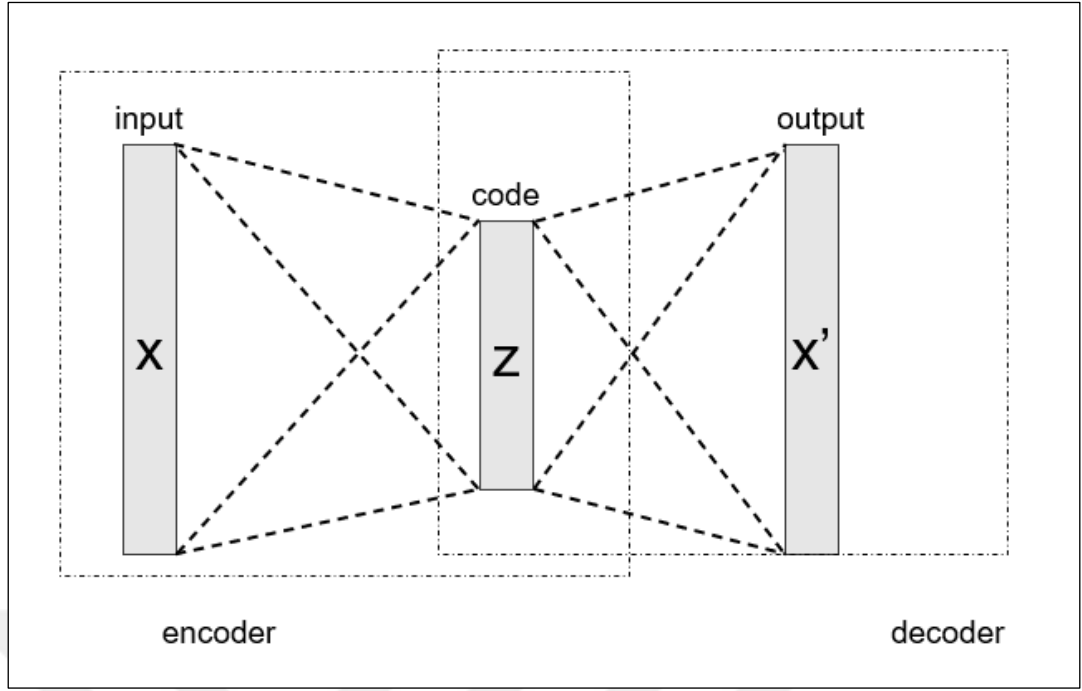


Figure 3.3: The basic scheme of a variational autoencoder.

The basic variational autoencoder scheme is shown in Figure 3.3. The model takes $\mathbf{x}$ as input. The encoder compresses it into the hidden field. The decoder takes as input the information sampled from the latent space and generates $\mathbf{x}'$ as like $\mathbf{x}$ as possible. In a generative domain, we are interested in expressing the probability of a data point $\mathbf{x}$ by marginalizing on a vector of latent variables:

$$P(X) = \int P(X|z)P(z)dz \approx E_{z \sim P(z)} P(X|z) \quad (3.9)$$

$p_{\theta}(\mathbf{x}|\mathbf{z})$ is expected to be close to zero for most values of $\mathbf{z}$, contributing insignificantly to the estimation of $p_{\theta}(\mathbf{x})$, making this type of sampling in the latent space practically impossible. The variational strategy takes advantage of sampling from an auxiliary "inference" distribution $q_{\phi}(\mathbf{z}|\mathbf{x})$, with the goal of obtaining $\mathbf{z}$ values that are more likely to contribute to the generation of $\mathbf{x}$. Variational autoencoder networks uses KL-divergence as loss function, KL-divergence is a way to measure dissimilarity between two probability distributions, based on information theory. The following equation describes the relationship between $p_{\theta}(\mathbf{x})$ and $E_{z \sim q_{\phi}(\mathbf{z}|\mathbf{x})} p_{\theta}(\mathbf{x}|\mathbf{z})$, where KL indicates the Kulback-Leibler divergence:

$$\log(P(X)) - KL(Q(z|X)||P(z|X)) = E_{z \sim Q(z|X)} \log(P(X|z) - KL(Q(z|X)||P(z))) \quad (3.10)$$

Since the KL-divergence is always positive, the term on the right provides a Lower Bound of Evidence (ELBO) for log-probability $\mathbf{P}(\mathbf{X})$. To make the ELBO formulation appropriate for training, the problem formulation and VAE structure must be somewhat modified [26]. The non-differentiable operation of stochastic sampling is used to sample from the latent space and feed the probabilistic decoder. The essential assumption regarding the latent space is that it may be thought of as a collection of multivariate Gaussian distributions, and hence as

$$z \sim q_{\Phi}(z|x) = \mathcal{N}(\mu, \sigma^2) \quad (3.11)$$

Given $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ defined as the element-wise product, the reparameterization trick modifies the above equation as

$$z = \mu + \sigma \odot \epsilon \quad (3.12)$$

The VAE becomes trainable because of this transformation (which can be extended to non-Gaussian distributions), and the probabilistic encoder must learn how to map a compressed representation of the input into the two latent vectors μ and σ , while stochasticity is kept out of the updating process and injected into the latent space as an external input via the random vector ϵ .

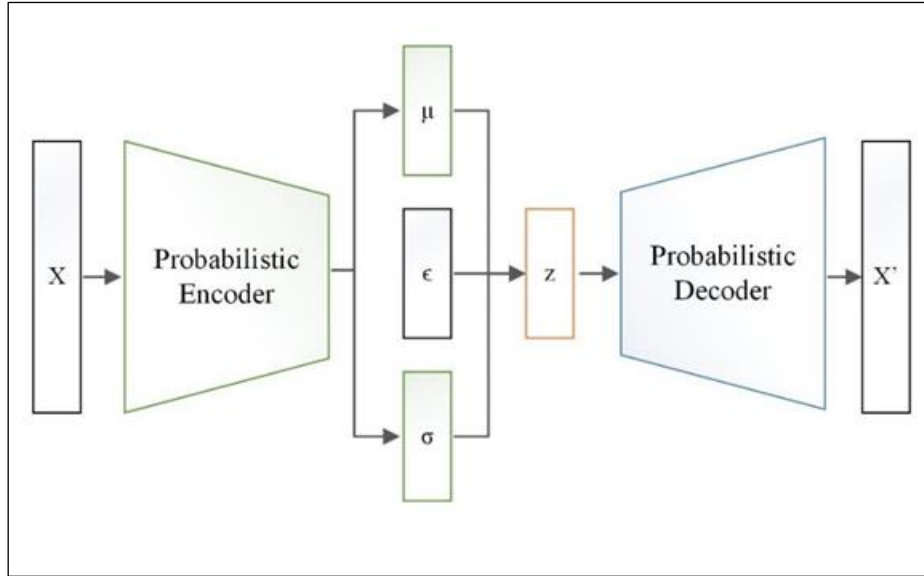


Figure 3.4: Scheme of a variational autoencoder after a reparameterization trick.

3.2.2. Conditional Variational Autoencoder

Suppose that, in addition to generating new digits, we wish to add digits to an existing string of digits written by a single individual. This is comparable to a real challenge in computer graphics known as hole filling: given an existing image from which an unpleasant item has been removed, the aim is to replace the hole with pixels that appear reasonable. Significantly complicating these difficulties is the multimodality of the space of feasible outputs: there are several choices for the next number or the extrapolation pixels. In this circumstance, a traditional regression model would fail since the training goal often penalizes the distance between a single prediction and the actual value. As it minimizes the predicted distance, the best answer the regressor can offer for such a situation is something in the middle of the options. In the case of numbers, this will most likely appear as a meaningless blur that represents the "average picture" of all conceivable digits and styles. We require an algorithm that, given a text or a picture, generates a complicated, multimodal distribution from which to sample. The conditional variational autoencoder (CVAE) [7, 8] changes the preceding section's mathematics by simply conditioning the whole generating process on an input. CVAEs enable us to solve issues with a one-to-many mapping between input and output without having to explicitly describe the output distribution structure.

The variational autoencoder, often known as the VAE, is a directed graphical generative model that has shown good results and is considered one of the most advanced approaches to generative modeling. It is assumed that the data is generated by a random process involving an unknown continuous random variable z . It is assumed that the z is derived from a prior distribution $p_{\theta}(z)$, and that the data is derived from a condition distribution $p_{\theta}(x|z)$, with $\mathbf{X}$ representing the data. The z is also referred to as the data $\mathbf{x}$'s hidden representation. It has an encoder and a decoder, much like any other autoencoder design. The encoder tries to learn $q_{\phi}(z|x)$, which is the same as learning a hidden representation of data $\mathbf{X}$ or encoding data $\mathbf{X}$ into a hidden representation (probabilistic encoder). The decoder tries to learn $p_{\theta}(x|z)$, which is responsible for decoding the hidden representation into input space. Figure 3.5 is a representation of the graphical model.

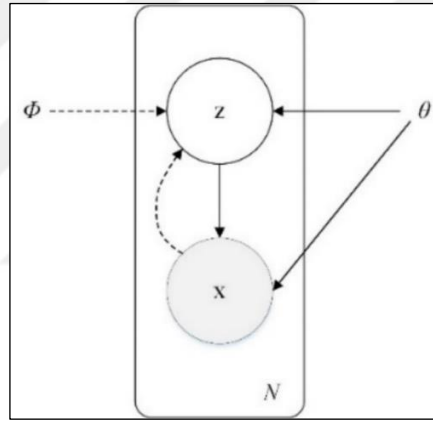


Figure 3.5: Graphical model of CVAE network.

In figure 3.5 the solid line represents the generative model $p_{\theta}(z) p_{\theta}(x|z)$, whereas the dashed line represents the variational approximation $q_{\phi}(z|x)$ to the intractable posterior $p_{\theta}(z|x)$. The variational parameters and the generative model parameters are simultaneously learnt.

The one issue with using VAE to generate data is that we have no control over the type of data it generates. For instance, if we train a VAE with the MNIST data set and then try to generate images by putting $Z \sim \mathcal{N}(0, I)$ into the decoder, we will get different random digits. Since we cannot control the label of the character to be produced in the VAE model, changes have been made on the VAE model. Let's assume that given an input y (label of image), we want our generative model to produce output

X (image). So, the VAE process is modified as follows: Given an observation y , z is drawn from the previous distribution $p_\theta(z|y)$, and the output x is generated from the distribution $p_\theta(x|z, y)$. Please note that for simple VAE the prior is $p_\theta(z)$ and the output is generated from $p_\theta(x|z)$. Encoder attempts to learn $q_\phi(z|x, y)$, which is identical to learning the hidden representation of data X or encoding X into the hidden representation conditioned on y . The decoder tries to learn $p_\theta(x|z, y)$, by decoding the hidden representation to input space when y is present. A visual representation of this model can be seen here in the figure 3.6 [33].

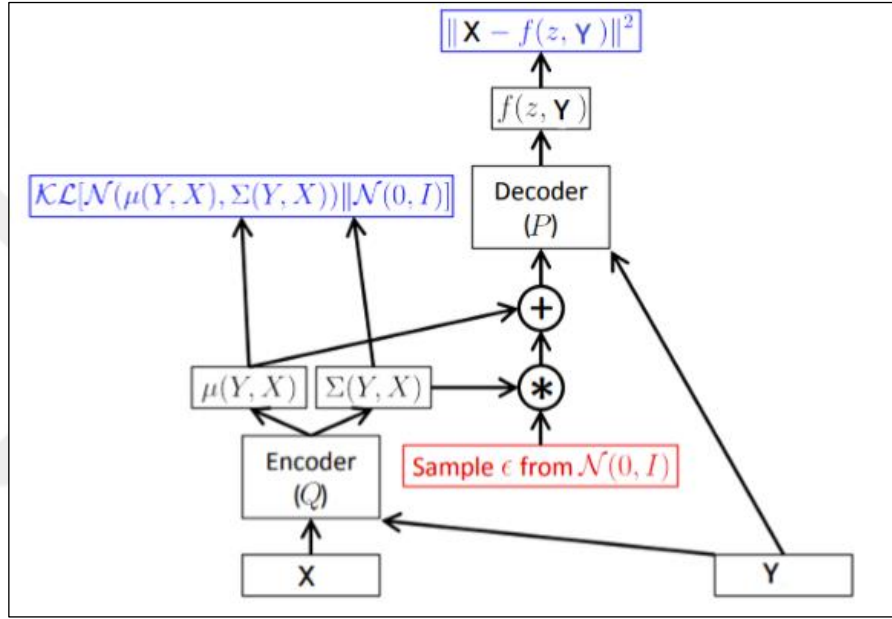


Figure 3.6: The basic scheme of a conditional variational autoencoder.

The network consists of two components: an encoder network E with parameters and a decoder D . $p_\theta(x|z)$ is a decoder from latent space to data space, whereas $q_\phi(z|x)$ is an encoder from data space to latent space. This optimization may alternatively be seen as reducing the reconstruction loss with the KL divergence serving as the regularizes. Conditional Variational Autoencoders (CVAE), which were described by [37] for the first time, maximize the variational lower limit of the conditional likelihood $p_\theta(x|y)$ in a similar fashion, hence enabling the generation of samples with certain desirable attributes (as encoded by y).

$$L(\Phi, \theta; x, y) = -KL(q_\phi(z|x, y) || p_\theta(z|y)) + E_{q_\phi(z|y)}[\log p_\theta(x|z, y)] \quad (3.13)$$

3.3. Proposed Architectures

In this study, two separate models including Conditional Variational Encoder model-based single-tasking and multi-tasking learning were tested for the style-based handwritten character generation problem. The architectures of the proposed methods are shown in Figure 3.7 and Figure 3.8. In the first proposed architecture, on the Conditional Variational Autoencoder network, the features of the handwritten character given as input are extracted by the Encoder network. A conditional variational autoencoder has an extra input for both encoder and decoder. This input is used to generate the requested character (image) as a condition. When the problem is formulated, let $\{X, Y, W, C\}$ be a multi-writer, handwritten character set, $W = [w_i]_{i=1}^N$, W is the set of authors and N represents the number of authors. C set are labels for handwritten digits, $= [c_t]_{t=0}^{10}$. The X and Y handwritten image sets created to express the input and output of each author are as follows; $[x_{w_i,k}]_{k=0}^K$, $x_k \in X$ and $[y_{w_i,m}]_{m=0}^M$, $y_m \in Y$, $x_k \notin Y$, $y_m \notin X$. By using the $G: X \rightarrow Y$ relation, when an image from the set X is given to the model as input with a condition from the set C , it is expected to generate the Y image. The structure of the developed model is shown in Figure 3.4. The encoder, designated as E in the figure, with the size of 128×128 given as input, passes through the network. The normal distribution is calculated on the feature vector extracted from the encoder network and creates the hidden space vector $\mathbf{z}$. The vector indicated as $\mathbf{c}$ in the figure is the condition vector and combined with the calculated vector $\mathbf{z}$, the decoder is given as input to the network. The decoder network generates 128×128 output based on the given character label.

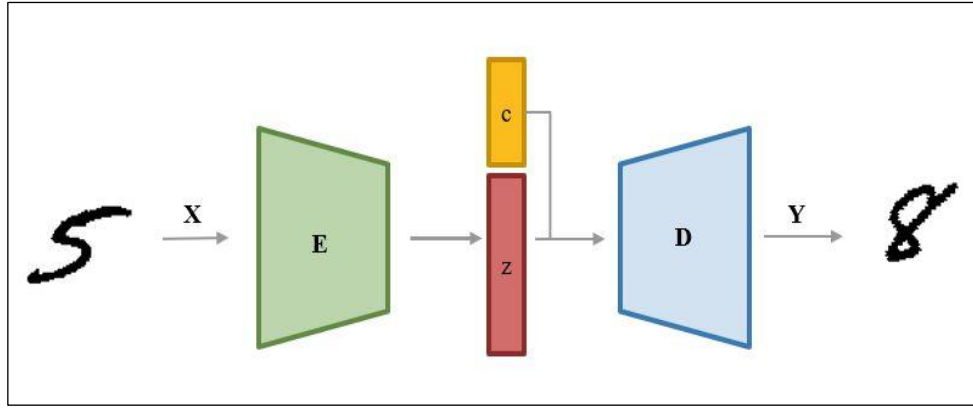


Figure 3.7: Architecture of the proposed conditional variational autoencoder network.

In the architecture shown in Figure 3.7, the network shown as **E** is the encoder network of the conditional variational autoencoder. In the proposed method, there are sequential convolutional layers in the encoder part. CNNs are a sort of method for deep learning used to handle data having a grid-like architecture. CNNs are a form of deep learning system used to handle spatially or temporally related data. CNNs are comparable to other neural networks, but their usage of a sequence of convolutional layers adds an additional layer of complexity. Convolutional layers are an important element of Convolutional Neural Networks (CNNs). A CNN has following 4 layers; input layer, convolution layer, pooling layer, fully connected layer. The input layer is the data that is fed into the CNN. In proposed architecture input consist of a 128-by-128-pixel grayscale picture. In contrast to FNN, we do not "flatten" the input to a 1D vector and instead send it to the network as a 128 x 128 matrix. This facilitates the capture of spatial connections.

Multiple filters constitute the convolution layer (also called kernels). 2D filters are applied to 2D images. Consider a grayscale picture of 28 pixels by 28 pixels. Each pixel is represented by a number between 0 and 255, where 0 represents black, 255 represents white, and the numbers in between indicate varying shades of gray. Suppose we have a 3 by 3 filter (9 values in total) whose values are randomly set to 0 or 1 at random. Convolution is the process of putting a 3 by 3 filter in the upper left corner of an image, multiplying the filter values by the pixel values and adding the results, then shifting the filter to the right by one pixel and repeating this process. When we reach the upper right corner of the picture, we slide the filter down one pixel and begin again

from the left. When we reach the lower right corner of the picture, this procedure will conclude.

Deep learning employs pooling layers, a kind of convolutional layer. Pooling layers minimize the spatial dimension of the input, making it simpler to process and less memory intensive. Pooling also reduces the number of parameters and makes training more efficient. Maximum pooling and average pooling are the two primary forms of pooling. Max pooling extracts the largest value from each feature map, while average pooling extracts the average. Typically, pooling layers are employed after convolutional layers to minimize the size of the input before it is given to a fully connected layer.

A CNN's last layer is a completely linked layer. We link all the preceding layer's nodes to this completely connected layer, which is responsible for image classification. A typical CNN contains many convolution layers and a pooling layer. Each convolution and pooling layer is accountable for extracting features at a distinct level of abstraction. For instance, the first layer's filters might identify horizontal, vertical, and diagonal edges. The filters in the subsequent layer were able to recognize forms, whereas the filters in the last layer were able to detect groups of shapes. Initialized at random, filter settings are learnt via a learning algorithm. This makes CNN very strong, since they can not only do classification but also perform feature extraction automatically. This separates CNN from other classification methods (such as Support Vector Machines) that cannot extract features. In the proposed method, the encoder network consists of the layer features shown in Table 1.

In the proposed model, we apply a multilayer CNN as the deconvolution-based decoder to perform the Transformation. With deconvolution, it is possible to convert a vector or a small matrix into a large matrix. In our model, deconvolution uses feature vectors after the encoder to generate 128x128 images to generate the target image. As the conjugate operation of convolution, deconvolution increases the

Table 3.1: The properties of the convolution layers of the encoder network in the proposed method.

Module	Specifications
Encoder	3×3 Conv-Norm-ReLU, 32 filters, stride 1
	3×3 Conv-Norm-ReLU, 64 filters, stride 2
	3×3 Conv-Norm-ReLU, 128 filters, stride 2
	3×3 Conv-Norm-ReLU, 128 filters, stride 2
	3×3 Conv-Norm-ReLU, 256 filters, stride 2

dimension of the input representation to the specified size matrix. Each of the L layers in the deconvolution contains f_l filters with a kernel size of k_l . The i^{th} filter $w_l^i \in \mathbb{R}^{k \times \dim}$ ($\dim$ defines the size of the description vector input) with stride s_l^i and padding p_l^i performs deconvolution on the input representation matrix $I \in \mathbb{R}^{m \times \dim}$ ($m \times \dim$ defines the size of the description vector input) the final hidden state of the encoder. The computation of convolution is illustrated as below equation 3.15:

$$c_l^i = g(W_l^i * X + b) \quad (3.15)$$

where X refers to the convolved matrix and g refers to non-linear activation function, which is ReLU [34] and deconvolution is its transposed operation. The deconvolution or transposed convolution operation is an up sampling that both up-samples feature maps and retains the connectivity pattern. Essentially, the deconvolutional layers increase and densify the input by performing convolution-like processes with numerous filters.

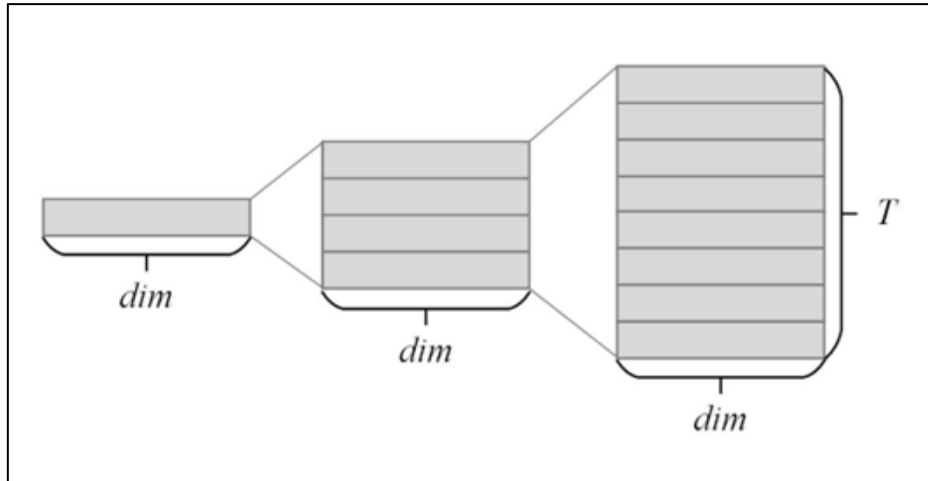


Figure 3.8: Deconvolution based decoder.

Compute the output shape of the direct convolution for a given input form, and then invert the input and output shapes for the transposed convolution.

In figure 3.9 based on deconvolution, is an illustration of a 1d deconvolution with a 2-dimensional input, a 4-dimensional kernel, a 1-dimensional padding, and a 2-dimensional stride. The depth refers to the width of the channel, which in our instance is narrow.

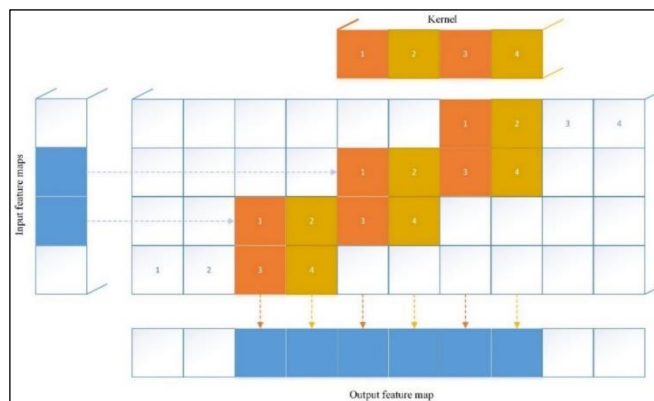


Figure 3.9: An illusion of the deconvolution process.

Table 3.2: The properties of the convolution layers of the decoder network in the proposed method.

Module	Specifications
Decoder	3×3 Conv-Norm-ReLU, 128 filters, stride 2
	3×3 Conv-Norm-ReLU, 128 filters, stride 2
	3×3 Conv-Norm-ReLU, 64 filters, stride 2
	3×3 Conv-Norm-ReLU, 32 filters, stride 2
	3×3 Conv-Norm-ReLU, 16 filters, stride 1

In the proposed method, the encoder also uses spline curves while extracting the styles of the network handwritten characters. While training the encoder model, we enable the network to extract features from spline curves with the multi-tasking learning method. The architectural figure of the network created using the multi-tasking learning method is as in 3.10.

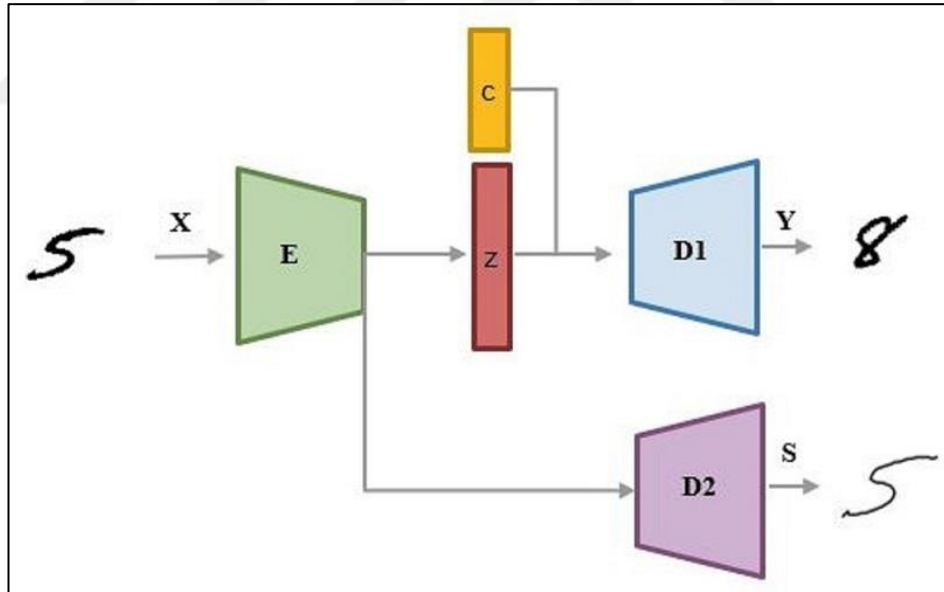


Figure 3.10: The proposed network including spline curves.

To summarize the proposed method, a handwritten character sample and the label of the desired character are given to the encoder network. The encoder network extracts the feature vector from the given handwritten image, then feeds the given models as $D1$ and $D2$ with reparameterization. The character to be produced is created with the

handwritten image and condition tag given in the $D1$ network. In the network shown as $D2$, representations of spline curves are created from the feature vectors coming from the encoder network.



4. EXPERIMENTS AND RESULTS

In this section, training and testing of the methods described in section 3 will be discussed. The operations on the NIST19 dataset used in the study were carried out with the OpenCV library. Deep learning models in the proposed method were created using pytorch version 1.8.0+cu111 [35]. The result of the models was visualized by using Matplotlib 3.5.1 [51]. The models were trained and tested by a system with NVIDIA GT 1650 graphics card with 4GB memory, AMD Ryzen 5 4600H CPU and 16GB RAM.

4.1. Dataset

There are two types of handwritten datasets, online and offline, for generating handwritten characters. Online datasets contain handwritten (x, y) coordinates and strokes. Offline datasets consist of handwritten images. In our problem, we use the NIST19 dataset, which is one of the offline datasets. The NIST19 dataset contains 128x128 numbers and alphabetic characters from full-page binary images of 3699 Handwriting Sample Forms (HSF). When creating NIST19, it was decomposed to include only class (character, number, uppercase and lowercase) and writer information. In our problem, handwritten characters have been labelled using deep learning methods, since the handwritten character dataset must have both the character label and the writer label. After labelling the data, some preprocessing steps were applied, and spline curves of handwritten character images were extracted. The processes are explained in detail in the subsections.

There are multiple instances of each character from a writer in the dataset. In Figure 4.1, examples of the numbers 5 and 6 of writers in the dataset are given. Due to the nature of handwriting, the characters in the figure are not the same, even if they belong to the same person.

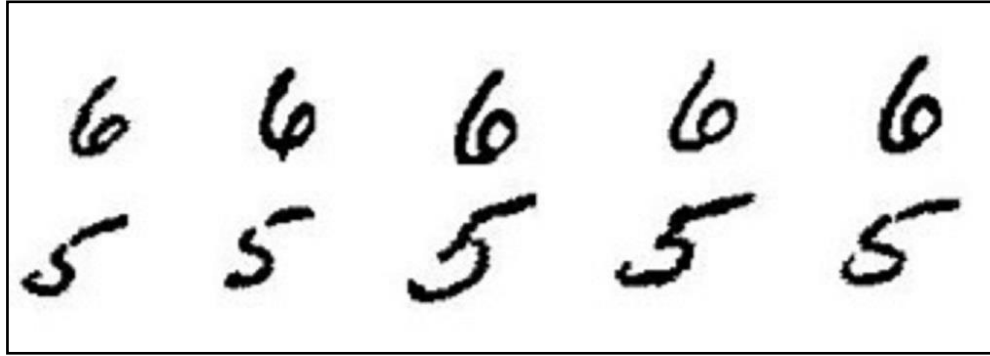


Figure 4.1: Example 5 and 6 characters from the NIST19 dataset by the same writer.

In the study on style-based handwritten character generation, our aim is to generate other characters of the writer that we have not seen yet from a limited number of characters, for example, since it is desired to generate other characters for a writer's 0 character, two sets, A and B, for training and likewise for testing. Two clusters, A and B, were created. An example of the training and test sets prepared is given in Figure 4.2.

Train A	Train B
2	6
5	9
8	4
1	7
3	0

Figure 4.2: Samples of the same author from the A and B sets created for the training set in the NIST 19 dataset.

In sets created for training and testing, for example, if train A has the number 0 belonging to the same writer, this character will not be found in train B set. A total of 2500 images from 50 writers in the NIST19 dataset were used in the experiments. 80% of these images were used for training, 10% for validation and the remaining 10% for testing. An average of 5 samples from each figure were used for each writer.

4.1.1. Pre-processing

In the NIST 19 dataset, handwritten characters are segmented to include some noise data. While calculating the spline curves of the characters, it was observed that the noise data also had a negative effect on the curves. A pre-processing is required without losing the style of handwritten characters. For this reason, as a method that does not take noise into account, the longest graph consisting of connected points that make up the handwritten character is calculated with the `connectedComponentsWithStats` [24] function in the OpenCV library. Sample character formed after the preprocessing step is shown in Figure 4.3.



Figure 4.3: From left to right, a character in the NIST19 dataset and a character formed after preprocessing.

4.1.2. Data Classification

In our problem, we need the writers of the handwritten characters and the information of the character label. The NIST19 dataset, on the other hand, does not contain character labels, so we used a CNN-based model for character classification. The neuroscience of the visual system is loosely inspired by convolutional neural

networks (CNNs). The basic concept is that each unit in a convolutional layer receives connections from a small number of units in the preceding layer (the unit's "visual field"), and these connections are organized in a two-dimensional topology to take advantage of spatial information. Each convolutional layer specifies a set of independent features to be learned, as well as the visual field size $N \times N$ (also called the kernel size). When compared to a network with fully connected layers, the units responsible for learning a feature share connection throughout the entire convolutional layer, resulting in considerably fewer network parameters.

The pooling layer, which uses subsampling to minimize the quantity of data flowing through the network, is another type of layer seen in CNNs. Each pooling unit takes input from a small number of units in the previous layer and applies a function to it (such as maximum or average). The overall consequence is that the information from the previous layer is coarser-grained, making the network less sensitive to minor changes in position. We will define a CNN for classification using three convolution layers with 5×5 kernels, each followed by a pooling layer with 2×2 kernels that calculates the maximum of their inputs. The first convolutional layer will learn 32 relatively low-level features, while the second and third layers will learn 64 high-level features. These features are then fed into a hidden layer (after flattening), followed by an output classification layer using softmax. The architecture of the network created is given in Figure 4.4.

Each convolution layer in the network includes a dropout setting of 25%. Dropout is a strategy for improving the ability of a network to generalize what it has learnt by making it less susceptible to noise and irrelevant correlations in the training data. Each training cycle, a randomly selected group of units in a dropout layer (here, 25% of the units) will be switched off (set to zero activation), with different random subsets being picked each cycle. Only during training does a unit drop out; once the network has trained, all units are involved in the classification of input data.

The proposed networks were built by using Keras [25] with TensorFlow 2.2.0 [26] backend. The CNN architecture trained and tested in Python.

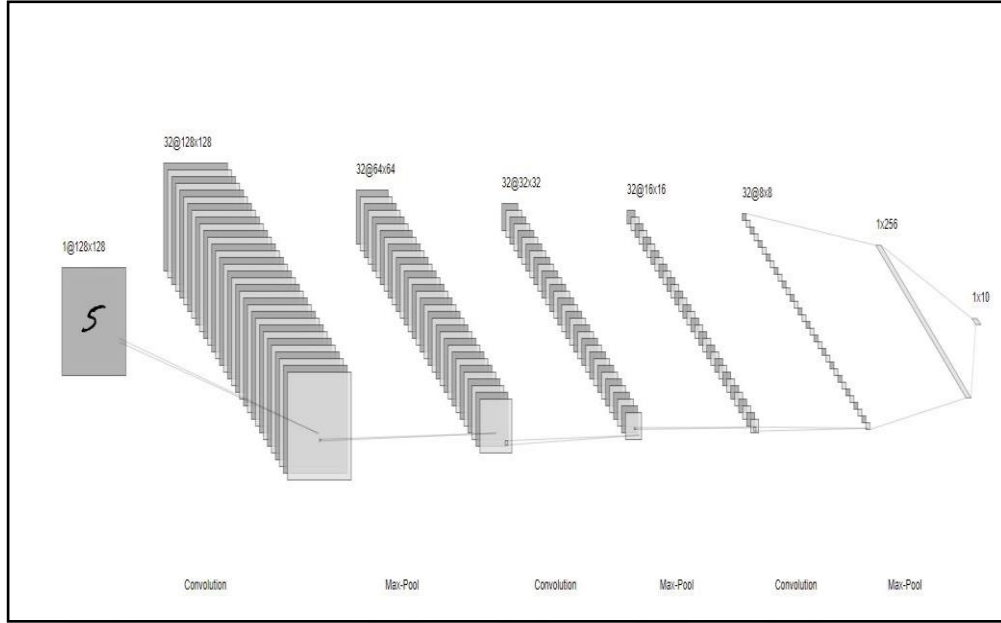


Figure 4.4: The architecture of the network used for character classification in the NIST19 dataset.

4.2. Single-Tasking Architecture

The CNN-based CVAE model in the proposed method is shown in figure 3.7. The Adam optimization function was used as the optimization function while the model was being trained. While the model was being trained, initial values were given to the hyperparameters and fine-tuning later. Table 4.1 defines the hyperparameter search space for the CVAE model shown in Figure 3.7. The optimization was initiated using the settings listed in Table 4.2, followed by up to 200 iterations with the predicted improvement acquisition function.

Table 4.1: The value range of hyperparameters.

Parameter	Type	Range
Learning Rate	Float	$[10^{-4}, 0.01]$
Number of Filters	Float	$[1, 1024]$
Filter Size	Odd Number	$[3, 25]$
Stride Size	Integer	$[1, 3]$

Initial hyperparameters and loss functions are given in Table 4.2. MSE and KL divergence were used for reconstruction loss as the loss function, and the default beta, epsilon and weight decay parameters were used in the Adam optimizer function, and the learning rate was formed with different values in the ranges given in table 4.1. The initial values used in the CVAE model created in Table 4.2 are given. After the model was trained with the relevant parameters, fine tuning on the parameters according to the loss values in the test phase and the images generated.

Table 4.2: Initial parameters for CNN based CVAE model.

Parameter	Value
Loss Function	MSE+KL Divergence
Optimizer	Adam
Training Epoch	100
Batch Size	32
Latent Space	50

The hyperparameters found by the optimizer are shown in Table 4.3. Various parameter configurations were manually tested and parameters were fine-tuned. Changed stride size, learning rate, dropout probability to best. In the created CVAE network, the latent space dimension was improved by the trials, and the tests were carried out with the value with the best result.

Table 4.3: Fine-tuned training parameters and values

Parameter	Value
Learning Rate	0.001
Training Epoch	100
Batch Size	128
Latent Space	80

The loss values during the training and validation processes of the CVAE network created with the single-tasking architecture are shown in figure 4.5. The graphs of the loss functions were considered while determining the epoch number.

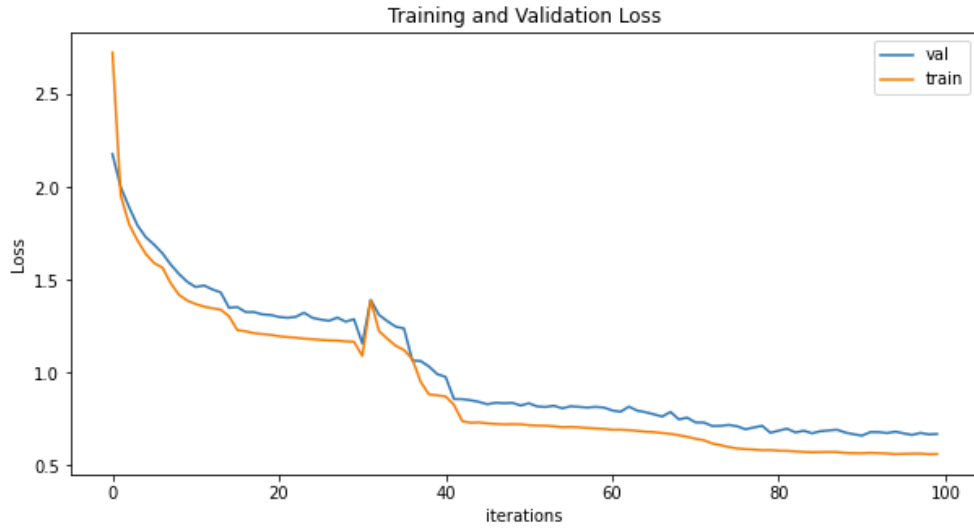


Figure 4.5: Loss values during training and validation.

Table 4.4 represents the CVAE model's testing outcomes. The X input field in the picture is the input provided to the model to extract the author style, while the X condition field is the one-hot vector of the label of the number wanted to be formed from the digit image provided as input.

Table 4.4: Test results of the CVAE model.

X_input	X_condition	Y_output	Y_groundtruth
3	[1 0 0 0 0 0 0 0 0 0]	0	0
5	[0 0 0 0 0 0 0 0 0 1]	9	9

4.3. Multi-Tasking Architecture

We conducted our evaluates on two different models for the problem of transferring a handwriting style from a limited number of characters belonging to a person. In the first model we developed, there is the learning part of the spline curves that we mentioned above. Afterwards, we saw that the results were better when we added the spline curves to the encoder part of the CVAE model. The structure of the

developed model is shown in Figure 3.10. The encoder, designated as E in the figure, with the 128×128 size, digit 5 given as input, passes through the network. The normal distribution is calculated on the feature vector extracted from the encoder network and creates the hidden space vector z . The vector indicated as c in the figure is the condition vector and combined with the calculated vector z , the solver $D1$ is given as input to the network. The decoder network produces 128×128 output based on the given character label. At the same time, the output of the encoder network is passed through the decoder network, which is shown as $D2$, which is created to learn the spline curves, and produces spline curves as output. The parameters used in the multi-tasking network that we created using spline curves in feature extraction are given in Table 4.5.

Table 4.5. Initial settings of the spline based CVAE network.

Parameter	Value
Loss Function	MSE+KL Divergence+MSE (spline)
Optimizer	Adam
Training Epoch	200
Batch Size	32
Latent Space	80

The loss function we used while training the model consists of three parts: The KL Divergence metric used to approximate the probability distributions of the encoder and solver networks given in equation (4.1), the mean square error between the expected and the produced output in equation (4.2), and equation (4.3). In the mean square error functions between the render results of spline curves and the expected spline curve are given. The weighted sum of the relevant functions forms the loss function (Equation (4.4)).

$$L_1(x, y; \theta, \phi) = -KL(q_\phi(z|x, y) || p_\theta(z|x)) \quad (4.1)$$

$$L_2(x, y) = \sum_{i=1}^D (x_i - y_i)^2 \quad (4.2)$$

$$L_3(s, r) = \sum_{i=1}^D (s_i - r_i)^2 \quad (4.3)$$

$$L = \alpha L_1 + \beta L_2 + \phi L_3 \quad (4.4)$$

The CVAE network with the loss equations given and the spline given in figure 3.10 is trained with multi-tasking learning with the parameters in table 4.15. During the training, experiments were made with different parameter sets until optimal results were obtained.

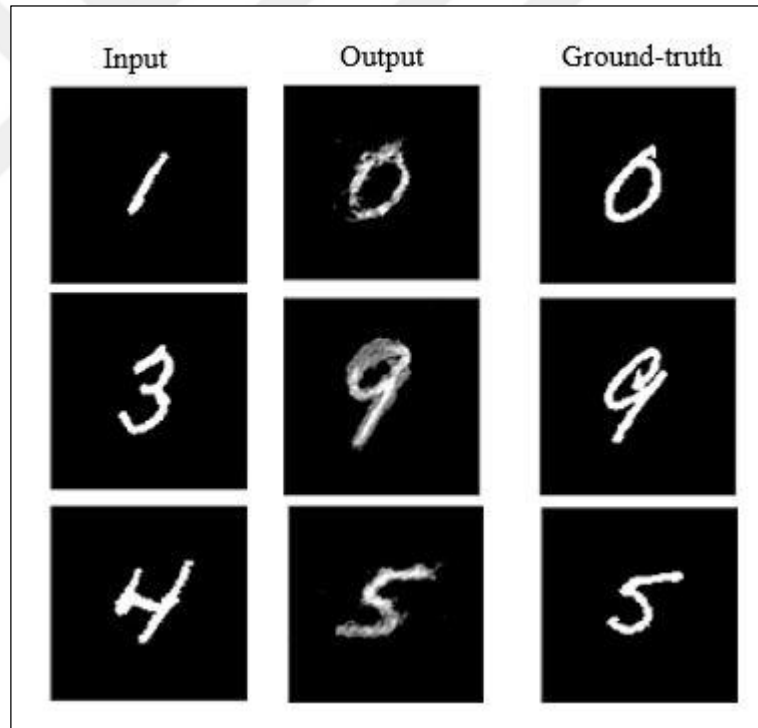


Figure 4.6: Generated CVAE model with different inputs from one author the outputs that happened.

Figure 4.6 shows the results we get when we input different figures from an author to the model. When we give the number 1 as the input and the label 0 as the condition, the result obtained is in the first line. The label of the number to be produced is given

to the model as a condition. Signal-to-noise ratio (PSNR) and structural similarity (SSIM) [15] metrics were used to measure the quality of the handwritten numbers produced. Calculated values for 3 different models are given in Table 4.16. PSNR generally gives the logarithm of the difference in pixel distortion (RMSE) in the entire image and may be insufficient to detect structural mismatches. Especially since the structural characteristics of handwritten numbers are more prominent, the structural similarity between the real and the produced character was also measured with the SSIM metric. According to the three models tested, the Conditional Variational Autoencoder (CVAE) with splines gave very good results compared to the other models.

Table 4.6: PSNR and SSIM of handwritten characters but the others.

Model	PSNR	SSIM
CVAE-Spline	17.8	0.78
CVAE	30.2	0.67
VAE	40.1	0.57

5. CONCLUSIONS

Handwriting generation is the process of making images that look like real handwriting. It can be used to improve text recognition systems, customize fonts, and solve CAPTCHAs, among other things. The main ways that handwriting synthesis works are by simulating shape and movement. Approaches to shape simulation look at how shapes in images are put together geometrically. Approaches to movement simulation try to come up with a model to explain how neuromuscular movements happen. Many works in the literature were used to gather important information about handwriting synthesis, such as input/output levels, data types, and script. Also, different ways to evaluate were talked about. It was said that both online and offline data showed promise for deep learning models. Still, it's hard to copy a writer's style. There are GAN-based models that can change a picture of a font into the style of a writer's handwriting. The conclusion is that it is hard to make a specific handwriting style from a small number of offline samples.

In this study, unlike the studies in the literature, spline curves created from the skeletons of the characters were used while generating handwritten characters. In the tests performed on the Latin alphabet to represent the style of a character, spline curves were found to be useful in style extraction. A multi-tasking learning network is proposed by using spline curves in the encoder network used for feature extraction of the Conditional variational Autoencoder (CVAE) model, which is one of the generative models. While working on the style-based handwritten character generation problem, tests were carried out on two different models. The first developed model consists of a conditional variational mesh that does not include spline curves. While investigating how the obtained results can be improved, it was thought that spline curves could be good for character representation. When the obtained results are compared, it is seen that the spline-based network gives better results than the first tested network.

An algorithm has been designed for the purpose of predicting how new numbers will be written by the author based on a restricted number of number samples that already belong to the author within the scope of this study. The solution to this issue is analogous to developing one's own writing style with a restricted set of handwritten characters. It was discovered that our system was able to accurately anticipate the

writing style of a certain author. We have high hopes that the model that we have built will be useful in solving challenges involving style-based picture translation. Our research appears to be the first of its kind in this field to make use of spline curves in autoencoders that generate number text, at least to the best of our knowledge. In our investigation, we relied solely on the author's original handwritten count.



REFERENCES

- [1] Graves A., (2013), “Generating Sequences With Recurrent Neural Networks”, arXiv: 1308.0850.
- [2] Fogel S., Averbuch-Elor H., Cohen S., Mazor, S., Litman, R., (2020), “Scrabblegan: Semi-supervised varying length handwritten text generation”, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 4323–4332, Virtual, June.
- [3] Sohn, K., Yan X., Lee H., (2015), “Learning structured output representation using deep conditional generative models”, Advances in Neural Information Processing Systems, 2(1), 3483–3491.
- [4] Chung, J., Kastner K., Dinh L., Goel K., Courville, A., Bengio, Y., (2015), “A recurrent latent variable model for sequential data”, Advances in Neural Information Processing Systems, 28(1), 2980–2988.
- [5] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., (2014), “Generative adversarial nets.”, Communications of the ACM, 63(11), 139-144.
- [6] Mirza M., Osindero S., (2014), “Conditional Generative Adversarial Nets.” arXiv preprint arXiv: 1411.1784.
- [7] Aksan E., Pece F., Hilliges O., (2018), “DeepWriting: Making digital ink editable via deep generative modeling”, Conference on Human Factors in Computing Systems - Proceedings, Montreal/Canada, April.
- [8] Kotani A., Tellex S., Tompkin J., (2020). “Generating Handwriting via Decoupled Style Descriptors”, European Conference on Computer Vision (ECCV), Glasgow/UK, August.
- [9] Aksan E., Hilliges O., (2019), “STCN: Stochastic temporal convolutional networks”, 7th International Conference on Learning Representations ICLR, New Orleans/Louisiana/United States, May.
- [10] Chang B., Zhang Q., Pan S., Meng L., (2018), “Generating Handwritten Chinese Characters Using CycleGAN”, IEEE Winter Conference on Applications of Computer Vision, 199–207, Lake Tahoe/United States, March.
- [11] Alonso E., Moysset B., Messina R., (2019), “Adversarial Generation of Handwritten Text Images Conditioned on Sequences.”, arXiv eprint arXiv: 1903.00277.
- [12] Haines T., Aodha O., Mac B., (2016), “My text in your handwriting”, ACM Transactions on Graphics, 35(3), 1-18.

- [13] Kang L., Riba P., Wang Y., Rusiñol M., Fornés A., Villegas M., (2020), “GANwriting: Content-Conditioned Generation of Styled Handwritten Word Image”, Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12368 LNCS, 273–289.
- [14] Bengio Y., (2009), “Learning deep architectures for AI”, Foundations and Trends in Machine Learning, 1312, University of Montreal, Canada.
- [15] Makhzani A., Frey B., (2014), “K-sparse Autoencoders”, 2nd International Conference on Learning Representations (ICLR), Banff/Canada, April.
- [16] Web 1, (2011), <https://web.stanford.edu/class/cs294a/sparseAutoencoder2011new.pdf>. (Date of Access: 17/09/2022)
- [17] Kingma D. P., Welling M., (2014), “Auto-encoding variational bayes.”, 2nd International Conference on Learning Representations (ICLR), Banff/Canada, April.
- [18] Rezende J., Mohamed S., Wierstra D., (2014), “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”, 31st International Conference on Machine Learning, 3057–3070, Beijing/China, June.
- [19] Kramer M. A., (1991), “Nonlinear principal component analysis using autoassociative neural networks”, AIChE Journal, 37(2), 233–243.
- [20] Abbasnejad M. E., Dick A., Van Den Hengel, A., (2017), “Infinite variational autoencoder for semi-supervised learning”, 30th IEEE Conference on Computer Vision and Pattern Recognition, 781–790, Honolulu/Hawaii, July.
- [21] Kameoka H., Li L., Inoue S., Makino S., (2019), “Supervised determined source separation with multichannel variational autoencoder”, In Neural Computation, 31(9), 1891–1914.
- [22] Doersch C., (2016), “Tutorial on Variational Autoencoders”, arXiv: 1606.05908.
- [23] Nair V., Hinton G. E., (2010), “Rectified linear units improve Restricted Boltzmann machines”, 27th International Conference on Machine Learning, 807–814, Wisconsin/US, June.
- [24] Web 2, (2022), <https://github.com/pytorch/pytorch> (Date of Access: 06/06/2022)
- [25] Web 3, (2022), https://docs.opencv.org/3.4/d3/dc0/group_imgprocshape.html (Date of Access: 02/05/2022).
- [26] Web 4, (2022), <https://github.com/keras-team/keras> (Date of Access: 05/05/2022).

- [27] Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Kudlur M., (2016), “TensorFlow: A System for Large-Scale Machine Learning”, In OSDI-16, 265-283.



BIOGRAPHY

Özdenur Uçar received her B.Sc degree in Computer Engineering from Kocaeli University at 2019. She started her professional carrier as Researcher, Software Engineer at TUBITAK BILGEM at 2019. Her research interests include computer vision, artificial intelligence, machine learning, deep learning, graph embedding and software architecture. Since 2019, she has been a graduate student at the Computer Engineering Department of Gebze Technical University.



APPENDICES

Appendix A: The Publications about the Thesis

Uçar Ö., Genç Y., (2022), “Style-based Handwritten Character Generation”, The 30th IEEE Conference On Signal Processing And Communications Applications (SIU), 1-4, Safranbolu, Türkiye, May.

