University of Basel

Papyri Fragment Style Study with Auto-Encoders

Master Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Center for Data Analytics

Examiner: Prof. Dr. Ivan Dokmanič
Supervisor: Prof. Dr. Ivan Dokmanič, Dr. Isabelle Marthot-Santaniello, Dr. Sébastien Boyer, Dr. Rodrigo Cerqueira Gonzalez Pena

Jannik Jaberg
jannik.jaberg@unibas.ch
2017-054-370

04.09.2022

Acknowledgments

I thank Prof. Dr. Ivan Dokmanić for the opportunity to work on the topic of auto-encoders in this thesis. Additionally, I thank Dr. Isabelle Marthot-Santaniello for providing the dataset that I used for this thesis. Furthermore, I thank Dr. Rodrigo Cerqueira Gonzalez Pena and Dr. Sebastien Boyer, from whom I received great guidance during the whole process of this Master's Thesis. Further, I thank all of the above for the valuable suggestions and inputs throughout the thesis. Furthermore, I thank Lars Fluri for the moral support and patience with me during the process of completing this thesis. Last but not least, I would like to thank Lars for proofreading and reviewing this thesis.

Abstract

Deep learning achieves outstanding results on tasks ranging from handwritten digit recognition to emotion classification from speech. This thesis explores deep learning as a tool to analyse handwriting style in ancient Greek papyri. We work with a dataset of letters segmented from papyri fragments. The task is complicated by challenges such as variations in grain between the different papyri, different sizes and resolutions of the letter cliplets, different acquisition technologies used to acquire the different parts of the dataset, and the great diversity of encountered writing styles and epochs. We study the potential of deep convolutional auto-encoders to disentangle the content (the letter identity) from "style". By style, we informally refer to a collection of attributes that includes the writer identity.

Table of Contents

1 Introduction

Since the revolutionary LeNet from Yann LeCun [3], classifying hand written numbers works to near perfection. Beyond simple recognition of digits, one could imagine identifying the writing's "provenance", which may entail writer and style classification. Many recent approaches use variants of convolutional neural networks to explore and identify the origin of ancient handwriting [4][5][15][23]. Alternatively, deep convolutional autoencoders may be used to characterize artistic style and even perform "style transfer": rendering one image with a style of another [12]. As an application, deep autoencoder-based approaches to binarization of ancient writings now outperform conventional approaches [4]. This leads to the idea of using auto-encoder models to analyze the style of ancient handwriting and ultimately determine its origin.

1.1 Dataset

The dataset studied in this thesis comprises 11855 cliplets of letters coming from ancient papyri fragments of the Iliad. Each cliplet is labeled with the character of the greek alphabet it represents, as well as the fragment ID which fragment it originates from. The labeling on the character representation we call character labels. The greek alphabet has 24 different letters, therefore we have 24 different character labels. The distribution of labels is rather heterogeneous, since some letters are more used than others in every language. As expected, the vowels a α, e ϵ, i ι, o o have are the most frequent characters with about 1200 samples for each of them. Further, we have commonly used letters with a sample size of 300 to 900 and rarely used characters such as xi ξ, zeta ζ and psi ψ with only 38, 34 and 15 samples.[1] This uneven distribution is also characteristic for the fragment labels. Fragments '60306', '60891', '60583' and '61073' with a sample count of 774, 753, 740, and 540 are the most common ones. For 30 fragments we have more than a 100 samples. The smallest number of samples per fragment is mere 12.[2] The resolution of these

[1] Exact sample sizes for the character labels can be found in A.1.1
[2] Exact sample count for the fragment labels can be found in A.1.2

cliplets ranges from 11 by 11 pixels to 300 by 300 pixels. Some of them are well readable where as others can have holes in the letters or be really hard to read due to faded ink. Also, the brightness of the samples can vary due to either the scanning process for the digitalisation or just the bare age of the fragment themselves. Having this as a premise, a suitable way to bring the differing cliplets into a uniform resolution and format is needed for further analysis. This is done with a pre-processing pipeline described afterwards in section 3.1.

1.2 Motivation

1.2.1 What is Style

Having described the dataset in Section , we can state our hypothesis. As the title of the thesis indicates, we want to study style. Therefore, first and foremost, we need to define what style means in the given context. Obviously, cliplets with the same character labels can be written in the same style, an alpha is always an alpha and shares the same or similar style features throughout the whole data about alpha. However, within the label class alpha, different styles will be present. For example, an upper case alpha A could be written in a more convex or concave manner. This would be exactly the style disentanglement we are searching for. Further, a different style feature or measurement can be drawn out of the fragment labels. We expect that cliplets that originate from the same fragment were written by the same author. Therefore, we have relationships that can be analyzed inter and intra character labels as well as fragment labels. For example, author A and B have the same looking upper case alpha A, therefore two fragment labels A and B show similar alphas. However, the style for their written upper case gamma Γ is totally different. Ideally, the fitted model should be able to recognize these details.

In addition, apart from the ground truth that we have from the labeling, some characters share similar style features. For example, alphas and deltas can be hard to distinguish without any textual context. Both share the triangular shape as their main feature. Further, even lambdas can be very close to alphas and deltas and not be distinguished by a nonprofessional. Below we listed potential main structural or base style features that characters can share. In the course of the analysis, new non-obvious main features may be discovered. These base style features can introduce ambiguity to or are a result of unsupervised learning, the chosen approach for a first analysis.

- alpha A, lambda Λ, delta Δ the triangular or hat feature, meaning having either three or two strokes triangularly placed.
- sigma σ, and omicron o share the "O" and "C" feature, meaning having one stroke in a round shape, either closed or opened on the right side.

This is very central for our hypothesis. We expect that the disentanglement of these shared style features could not be possible. However, the best outcome is a disentanglement of the style for

every character label. Even better would be an even more precise style disentanglement within a class, so that the model can distinguish between differently styled characters within a single character label class.

1.2.2 Auto-Encoders

An auto-encoder is a self verifying neural network. Essentially, an auto-encoder takes an input, compresses it and decompresses it again which forms the output. The learning takes place by comparing the original input and the reconstructed output. Therefore, an auto-encoder learns in an unsupervised manner by minimizing the loss between input and output. This is extremely powerful since the auto-encoder does not need any labeling. By making the compression very small, an auto-encoder is able to learn its own labeling of the data. This labeling does not have to be obvious when looking at the data. Therefore the auto-encoder manages to find hidden labels. However, if labeling of the data is present, we can augment the auto-encoder so it also takes labeling into account. The loss is now calculated by comparing the original input , the reconstructed output and the learned label. In addition, outlier detection is very easy with auto-encoders. Compared to traditional neural networks, auto-encoders also try to learn a representation for the whole input. Outliers that do not follow the majority of the data's properties have a bad reconstruction. Therefore the reconstruction error is a very good indicator for anomalies [1][20].

Having this short explanation makes the choice to use auto-encoders obvious. We are exactly searching for these hidden labels that hopefully represent different style labels. In addition, we have labeling already present to control and supervise the training if the hidden labeling is ambiguous.

1.3 Outline

The following paragraph briefly explains the structure of this thesis. This introduction to the dataset, hypothesis and choice of network architecture is followed by the chapter about related work and a short literature review. Within the chapter, research about writer identification, style disentanglement and use of neural networks and auto-encoders are examined. After that, we have a look at the first approach on the analysis with a convolutional auto-encoder. In a second approach, we augment the architecture with residual learning and draw further analysis of the latent space. In a last approach we condition the latent space of our residual auto-encoder architecture. Finally, this thesis is closed with the chapters Future Work and Conclusion. In which the work is reviewed, open questions get answered and an outlook for possible additions and improvements is given.

2

Writer Identification and Neural Networks

There already exist approaches for automatic recognition of patterns in data, for example to classify it. When looking at the state of the art Optical Character Recognition (OCR) for typewritten documents, these approaches are significantly faster than pattern recognition by humans. In return, applying OCR to handwritten texts is far more challenging. Size and style of handwriting change between authors. Even texts written by the same author can have great differences in handwriting style.
Nevertheless, writer identification is an actively researched and academically challenging field. Nasir et al. [15] state: "Though significant research endeavors have been made to address the writer identification problem in contemporary handwriting, the problem remains challenging when it comes to historical manuscripts primarily due to the degradation of documents over time." Algorithms analyzing historical handwriting are often exposed to the random noise caused by the nature of the document, such as stains or holes, ink fading, bleeding through or degrading. Desired are methods which can distinguish meaningful features and ignore these distortions or use said the distortions to their advantage. Furthermore, the field of style disentanglement [12] may be considered as well. Similar problems are solved there by extracting features of paintings to apply the style of a painting to a real world photograph and generate an artificial painting of said photograph in the style of the analyzed artist. The method used there are auto-encoders. In another paper auto-encoders are used to binarize handwritten texts [4]. We will see throughout the proposal that binarization is omnipresent in the field of pattern recognition. These made approaches lead to the idea of directly using auto-encoder models to analyze the style of ancient handwriting and make a guess of its origin.

2.1 Writer Characterization and Identification from Handwriting

A common approach for writer identification or characterization are convolutional neural networks (CNN). Nasir et al. [15] fed pre-processed images with annotated key points into a CNN

to extract features. Since the dataset they used was sparse, they introduced a two-step fine-tuning. Prior to the feature learning, different pre-processing techniques were researched. Since the images they studied contain a vast variation of papyri fiber backgrounds, it is crucial to extract foreground (letter) from background (papyrus). Investigated are adaptive binarization, edge detection, edge detection on binarization and deep Otsu binarization. Afterwards, keypoint detection is applied to the binarized images. By using the keypoints as centers, patches around them are extracted, preserving important writer-specific writing strokes. From there on they fine tuned their pre-trained network on the features of handwriting.

When looking into different approaches to writer identification, the academic consensus is that CNN architecture is the state of the art methodology [18][23][5]. Rehman et al. [18] use a deep CNN architecture, based on the AlexNet. Over 8 layers, 290400 dimensions are convoluted into 8192 and then the writers identity is retrieved. An SVM classifier after each layer measures the layers current accuracy on identification. As expected, the accuracy increases with each layer. The highest accuracy realized was by using the fifth convolutional layer as a freeze layer. Freezing means that the layer will not be trained and its weights will therefore not be changed. There, accuracy reached up to 92% on writer identification. In contrast to the approach of Rehman et al. [18], unsupervised approaches with CNN's are used because the lack of labeled data. Christlein et al. [5] provide a simple method for feature learning using deep neural networks without the need of labeled data. Their pipeline is constructed as follows: In a first step, scale-invariant feature transform (SIFT) keypoints [13] are extracted. From there, at each keypoint the resulting SIFT descriptor and a 32×32 patch of the image is extracted. The image patches serve as training input for a ResNet, whereas the SIFT descriptors are clustered and therefore serve as targets. This architecture provides artificial labels for the patches and generates data which makes feature learning possible. SIFT keypoints are suited very well for this task since as Lowe [13] state: "The keypoints have been shown to be invariant to image rotation and scale and robust across a substantial range of affine distortion, addition of noise, and change in illumination." Since handwriting most definitely show these impurities a reliable and trustworthy feature extraction without labels is possible. In addition, Christlein et al. [5] also state: " [...] the activation features trained without supervision are superior to descriptors of state-of-the-art writer identification methods." Since data labels often are missing, unsupervised learning out performing state-of-the-art methods is a premise that is favoured by data scientists and analysts. Having the data explain itself by the help of some algorithm such as generating SIFT keypoints, omits the labels or teacher and therefore the data teaches itself.
Further, during testing Christlein et al. [5] found out that their approach works better on binarized images, rather than gray scale. Binarization is a common pre-processing step taken to refine handwritings [4], which cancels out background noise in order to have higher contrast between fore- and background. Later in the proposal, binarization is revisited, due to its immense

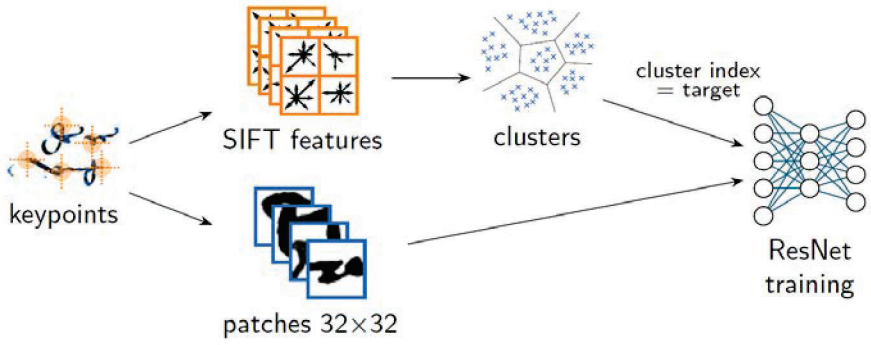


Figure 2.1: Architecture of Christlein et al. [5]

importance in the field of pattern recognition.

Another deep CNN approach is made by Xing and Qiao [23]. Their paper proposes: "DeepWriter, a deep multi-stream CNN to learn deep powerful representation for recognizing writers." More specifically, they state that writers can be recognised by capturing specific characteristics of their handwriting. These characteristics differ vastly from other writers. Again, a deep CNN architecture to extract these characteristics as features is proposed, since deep CNNs have demonstrated their effectiveness by improving state-of-the-art results with a large margin. DeepWriter is trained with patches of hand written texts with softmax loss on identification. Afterwards, the paper introduces data augmentation to boost performance. Lastly, Xing and Qiao [23] introduce a patch scanning strategy to operate on varied dimensions of hand written images. The results speak for themselves, lowest identification rate sitting at 93% and up to 99% accuracy in the best case, outperforming previous state-of-the-art approaches.

As we can see, broad research on writer identification has been already made. Most of the approaches include convolutional neural network architectures almost guaranteed. CNN architecture only by itself will not do the trick. Due to heterogeneous data, missing labels and other distortions, combining algorithms and augmenting CNN architectures is key for successful results.

2.2 Artistic Style Transfer

Based on these handwriting methods, we can see that feature learning plays the major role. These learned features are actually the style of how an author writes. From there we can draw the arc to style transfer. In the paper *Content and Style Disentanglement for Artistic Style Transfer*, Kotovenko et al. [12] describe an approach to gather stylistic features from images of artists and apply them to photographs, resulting in a artistic version of the photograph in the style of the artist. This is achieved by using auto-encoders, where two content images are fed into the content encoder and two style images are fed into the style encoder. Afterwards, all

possible pairs of content and style representations are generated using the decoder.

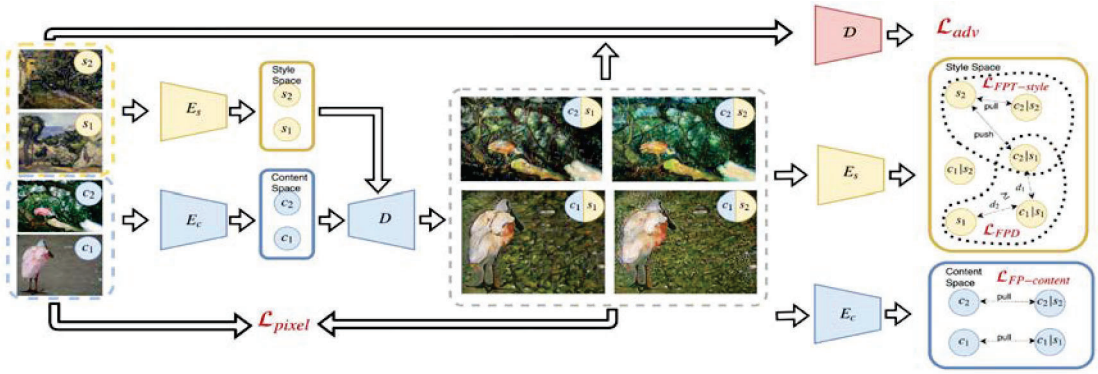


Figure 2.2: Auto-encoder architecture of Kotovenko et al. [12]

The encoders manage to synthesise the key features of the style images and therefore provides a representation of the style which should be applied to a photograph. This powerful auto-encoder architecture sparks the idea to analyze handwriting style and identify writes with auto-encoders rather than with the "conventional" (deep) CNN architecture.

2.3 Auto-Encoders for Binarization

As broadly described already, nearly any dataset goes through a pre-processing step. In handwriting analysis, a common pre-processing step is to already enhance the letters and lower the background noise. One way to do this is to binarize the images before feeding them into the classification or analysis algorithm [15][5]. Binarization can be achieved in many ways: Thresholding, Adaptive Thresholding or Otsu Binarization just to name a few [3].

In the paper *A selectional auto-encoder approach for document image binarization* [4] convolutional auto-encoders are used to achieve binarization on handwriting. Calvo-Zaragoza and Gallego [4] state: "In this paper we discuss the use of convolutional auto-encoders devoted to learning an end-to-end map from an input image to its selectional output, in which activations indicate the likelihood of pixels to be either foreground or background. [...] This approach has proven to outperform existing binarization strategies in a number of document types." Again, having this promising feature detection which leads to "extracting a letter", the idea to use auto-encoders directly for style analysis or writer retrieval seems reasonable.

[3] OpenCV

3

Convolutional Auto-Encoder for Reconstructing Handwritten Ancient Greek

This chapter is dedicated to get a first interaction with the dataset. We analyze different pre-processing steps that can be made to bring the heterogeneous dataset in a uniform format, suitable for neural network training. Afterwards we give an overview on how auto-encoders, our choice of technique, work. Finally, we analyze and evaluate the results of the first auto-encoder implementation.

3.1 Pre-Processing

As anticipated, a well structured pre-processing pipeline is key for feature extraction. Since most datasets are not in an uniform format, data cleaning is necessary. As explained in subsection 1.2.1, we have to deal with a heterogenous variety of resolutions, as well as poor to good quality of the images. Therefore, after resolving cropping and scaling issues, an additional step is required to reduce dimensionality of the color channels. Reduction of dimensionality can be achieved by either grayscale or even binarization. This helps to perceive the information in each cliplet but reduce the complexity for a machine learning all three color channels. In fact, process testing lead to a switching of the approaches: In a first step reduce the color channels and in a second step, crop and scale. This leads to less data loss and better results.

Thresholding and Binarization

Thresholding and binarization is a tool in digital image processing for segmenting images. It transforms a grayscale image into a binary image, meaning only pure black and white pixels encode the image. A great tool to perform such thresholding and binarization is OpenCV[4],

[4] OpenCV

which was used to segment our images.

Adaptive Thresholding

Adaptive thresholding determines the value of a pixel based on a small region around it. This results in different thresholds in different regions of an image. This is useful for varying illumination or if an image is in general brighter or darker.

(a) (b)

Figure 3.1: Original alpha α in 3.1a and alpha α after adaptive thresholding in 3.1b

(a) (b)

Figure 3.2: Original epsilon ϵ in 3.2a and epsilon ϵ after adaptive thresholding in 3.2b

Otsu's Binarization

The Otsu method is of the most popular image binarization algorithms. It converts a grayscale image to a monochrome image. More precise, Otsu's method iterates over all possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold. It aims to have minimal spread between the two classes.

(a) (b)

Figure 3.3: Original alpha α in 3.3a and alpha α after Otsu's binarization in 3.3b

(a) (b)

Figure 3.4: Original epsilon ϵ in 3.4a and epsilon ϵ after Otsu's binarization in 3.4b

Cropping

After reducing the RGB color channels to a single one, cropping takes place. Since neural networks or auto-encoders can only process a uniform input format, resizing of the images is crucial. As a easy way of cropping, cropping only consists of cutting of excess pixels if the height exceeds the width or the other way round. Having this approach, minor data loss is inevitable. At this point this minor data loss is still forgiving, yet for the future pipeline to be avoided. A better way to crop excess pixels is to first calculate the center of mass of the pixels and create the largest possible square around the said center of mass. This results in only cropping the "most unnecessary" pixels.

Padding

Since cropping always emits data loss, an approach without data loss is desired. Padding is a different approach to resize images. By adding white pixels to the border, the shape of an image is changed but all data preserved.

Scaling

After cropping, the images have to be scaled to a uniform resolution. We chose 28×28 pixels, this is enough to still have information about the features in the cliplet, and not too big that the calculation expenses sky rocket. Here as well, scaling with interpolation introduces data loss. Again, at this point, data loss can be overseen but has do be taken into account later on. Also for scaling, padding can be an option. By searching the biggest resolution contained in the dataset, all other images can be padded to this resolution, resulting in no data loss.

3.2 Methodology

As it has been mentioned before, the main analysis and feature extraction of the cliplets will be done with auto-encoders. For the first time, auto-encoders were introduced by Hinton in the 1980s [19]. Since then, auto-encoders play a significant role in unsupervised learning as well as in deep architectures for transfer learning. The novelty introduced by auto-encoders was

"backpropagation without a teacher". By using the input data itself as teacher the addressed problem was solved [2]. From there on, auto-encoder research on architecture and learning approaches keep expanding.

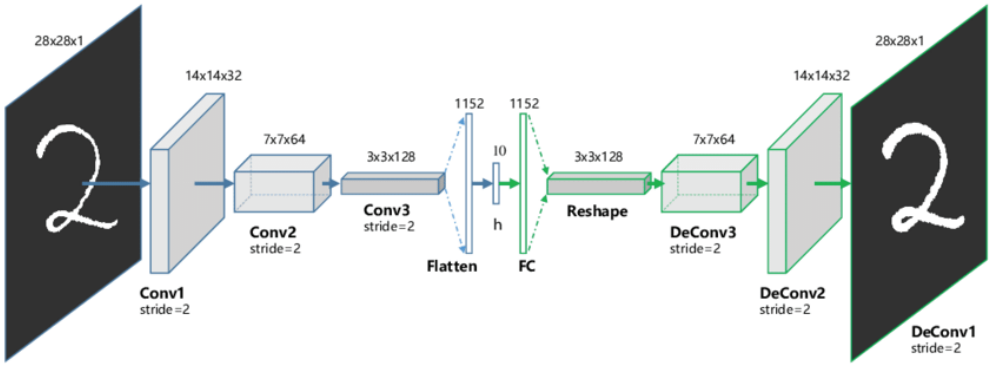


Figure 3.5: Example architecture of a convolutional auto-encoder
Source: analyticsindiamag.com

3.2.1 Auto-Encoder (AE)

An auto-encoder [2] contains two separate parts, the encoder and the decoder. The encoder learns the features of the input and maps it to the code. The code is the middle part, which contains some abstract representation of the data and holds the key features. The code or representation lays in the latent space parameterized by the last layer of the encoder. From this code, the decoder reconstructs the input. The encoder and decoder can be any sort of neural network, for example multi-layer perceptron, RNN or CNN. Since CNN architectures have been proven to be the most reliable and versatile neural networks for image analysis, it makes sense to incorporate different architectures of convolutional neural networks.

3.3 Implementation

Now, having the information needed on auto-encoders, the architecture can be chosen. We chose a simple convolutional neural network as our encoder, where as the decoder is the exact counter part which up-samples the encoded date the way the encoder encoded it. In detail the archtecture is as follows: The convolutional auto-encoder (convAE) consists of 3 convolutional layers, as we can see in Figure 3.6. In between layers, maxpooling reduces the dimensionality to a quarter of its preceding size. The input layer is an $1 \times 28 \times 28$ image. In the first convolutional layer the "depth" dimension gets expanded to 16 output dimension convoluted by 5×5 kernels with stride one. To avoid data loss, we also introduce a padding of size two, therefore the output dimensions height and width stay the same. Afterwards, a maxpooling layer with kernel size 2×2 and stride 2 reduces the height and width dimension by half. The next convolutional layer and maxpooling

layer follow the same rules but expand the depth by factor two. This results in a final dimension of $32 \times 7 \times 7$. In a last convolutional layer, this time with a 3×3 kernel with stride 1 and padding 2, the final dimensionality is convoluted to $4 \times 7 \times 7$, resulting in a 196 dimensional latent space.

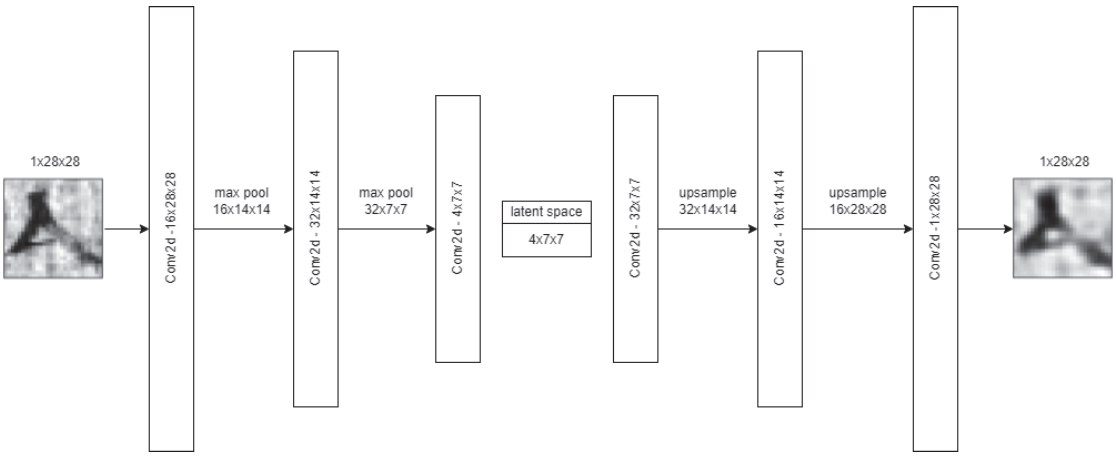


Figure 3.6: Architecture for the ConvAE.

The decoder is by definition the inverse of the encoder. As we can see in Figure 3.6, the decoder samples the height and width up to original size using and reducing the expanded depth dimension. More details for the architecture of the convolutional auto-encoder can be found in the GitHub repository under `autoencoders/convautoencoder.py` [5]

3.4 Evaluation

Testing the convAE with both pre-processing approaches, gray scale and binarization, lead to promising results. In Figure 3.7 we see the inputs in gray scale in the upper row and the decoded outputs of the convAE in the lower row. The blur in the second row indicates that in fact the auto-encoder is able to learn features from the input, since the characters reconstructed are clearly recognizable. Also, in Figure 3.8a and 3.8b we can see the produced $4 \times 7 \times 7$ code of the auto-encoder. When having a closer look at the feature maps, we can still see the remaining of the original input.

[5] https://github.com/cptunderground/cvae-papyri/blob/main/autoencoders/convautoencoder.py

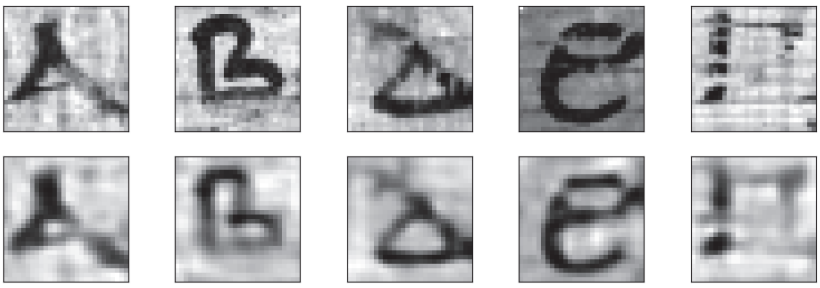


Figure 3.7: The letters alpha α, beta β, delta δ, epsilon ϵ and gamma γ above in gray scale and below in the decoded output.

(a) α (b) ϵ

Figure 3.8: The four feature maps of a gray scale alpha α and epsilon ϵ.

Also the binarized pre-processing was fed into the auto-encoder. In Figure 3.9 again input and output are matched. Here the learning can be seen even better. For example the artifacts from binarization in the letter beta vanish in the decoded image of beta. This means through the feature learning the auto encoder is able to identify that these artifacts only belong to a minimal way to the image. Also, when looking at gamma we can see that the vertical gaps in the letter get filled.

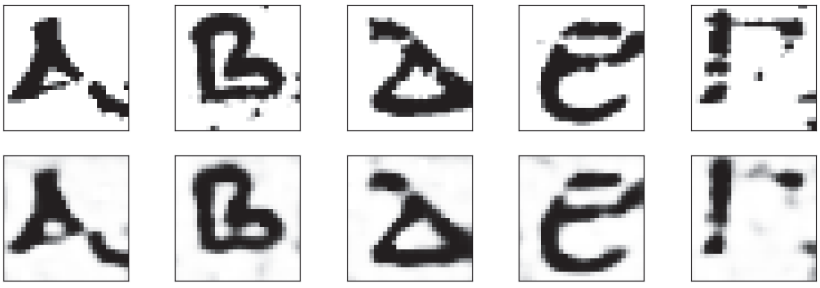


Figure 3.9: The letters alpha α, beta β, delta δ, epsilon ϵ and gamma γ above in Otsu binarization and below in the decoded output.

(a) α (b) ϵ

Figure 3.10: The four feature maps of a binarized alpha α and epsilon ϵ.

As said, the results are promising but not even anywhere to make a sophisticated guess about what was really learned. This first approach was a good exercise to get familiar with the dataset and its advantages and also drawbacks. Further, a first get to know with auto-encoders took place. The understanding, why it is the preferred choice as general architecture, was made and stated. In the next chapter we alternate the pre-processing and also augment the architecture of the auto-encoder to have a more powerful and robust machine analyzing the data, instead of fine-tune this first approach.

4

Elevated Residual Architecture for Auto-Encoders

This chapter discusses the augmentations made to the previous model. By switching up the pre-processing pipeline to have less to no data loss on the cliplets and choosing a higher resolution for the training data, the input for the model is much more descriptive than before. Further, the 3-layer convolutional architecture gets replaced by four-layer custom residual architecture. This architecture is much more suitable for this kind of image analysis and feature extraction. After having a close look at the actual architecture, we introduce three commonly know latent space analysis techniques. The resulting model is then analyzed with these techniques an first behaviors about style are made.

4.1 Methodology

In the last years the ResNet[9] has proven to be the state of art for pattern recognition and image analysis. With its special skip connection or shortcut architecture, the ResNet avoids problems that have to be faced in image analysis such as vanishing gradients. Naturally, using this kind of network to make our model better is obvious.

4.1.1 Residual Learning

As we know, neural networks can approximate every function. The accuracy of the approximation increases with each layer added to the network. But still, there is a limit of improvement. Actually, increasing a network's depth is not as simple as just stacking layers. Deep networks are hard to train than shallow networks. This is because of the vanishing gradient problem. As the gradient is back-propagated to previous layers, the result of the repeated multiplication of back propagation may make the gradient infinitely small. Therefore, as the network's depth increases, the performance is saturated and can even start degrading rapidly. This means, at some point adding layers will decrease accuracy. This is happen because of the stated problem such as vanishing gradients[9]. Therefore, a neural network with sufficient deep layer may not

even be able to learn one of the simplest functions such as the identity function. In some cases shallower networks perform better than their deeper counterparts. This is counter-intuitive to the statement that "adding layers increases accuracy" stated above. Yet, this is seen often in practice and is known as the degradation problem[9]. The core idea of the ResNet are the residual blocks. They introduce an "identity shortcut connection" which skips one ore more layers. This residual block is one way to work against the degradation problem caused by vanishing gradients.

4.1.2 Residual Block

Understanding the residual block is easy. As we know, conventional neural network layers feed directly into the next layers. In contrary, in a residual network each layer feeds into the next layer as well as into a layer 2 to 3 hops deeper in the network. Yet, what is the intuition to have an architecture like that, since understanding it is easy and straight forward. Before we take a closer look why these residual blocks are used, a residual block is shown below.

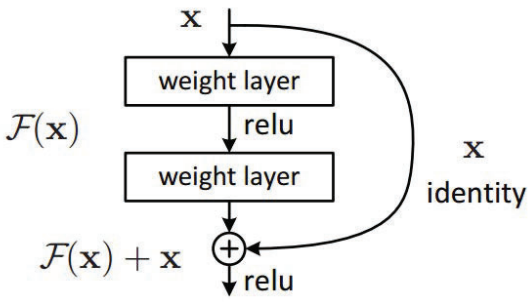


Figure 4.1: Single Residual Block [9].

By skipping layers and matching the accuracy of the shallow sub-network, we can imitate shallow networks. If we have a look at the residual block above, we can rely on the skip connection only to learn the identity function. So now, why is it called residual? Let us consider a layer which has input x and it needs to learn the distribution $H(x)$. Therefore, we get: $R(x) = Output - Input = H(x) - x$ and after rearranging it we get: $H(x) = R(x) + x$. Overall, the residual block tries to learn the true output, $H(x)$. As shown in Figure 4.1, due to the identity connection or shortcut the layers actually try to learn the residual $R(x)$. To summarize, traditionally a network learns the true output, whereas in a residual network it is actually the residual $R(X)$, hence the name: Residual Block.

4.2 Implementation

As stated in subsection 4.1.1 about residual learning, introducing residual blocks in a neural network and therefore also in an auto-encoder, gives us the possibility to build deeper networks without having to worry about vanishing gradients too much. This is exceptionally useful for

the analysis of our dataset since the resolution can be up to 300×300 pixels. Therefore, now we have the possibility to freely choose a big input dimensionality and still be able to sample it down to a fairly small latent space in the AE. This is done by just stacking residual blocks.

4.2.1 Pre-Processing Adjustments

In the first approach our pre-processing pipeline had many factors included that lead to information loss. In the second approach the idea is to minimize the data loss to an acceptable extend. In contrary to the first approach, where we set on cropping, this approach uses padding to get the cliplets into uniform pixel ratio. To achieve this we wrote a custom padding class that takes the images as input and returns them padded to a square resolution. For example sample s has the resolution of 20 by 30 pixels. The padded sample s' will have a resolution of 30 by 30 pixels, where the newly added pixels are just white ones. Now having the individual square resolution for each cliplet, the cliplets get resized to 64 by 64 pixels. The increase in pixels is made to preserve more information. The last decision that has to be made was if we go with or against the binarization approach. For the sake of simplicity we decided to first only look at the gray-scale approach and eventually incorporate the binarization at a later point. This architecture for the pre-processing pipeline gives us the possibility to scale the resolution of the cliplets as desired. Further, in the next section we can see how the network can deal with altering resolution choices with the aid of the ResNet[9] architecture.

4.2.2 Model Architecture

As discussed, the ResNet architecture allows us to stack layers and form deep networks without worrying about vanishing gradients and decreasing accuracy. In the section we have a detailed look at the used implementation of the ResNet. Actually, to explain the used architecture it is more comprehensive if we build it up from inside out. We have seen the theoretical abstraction of a residual block in Figure 4.1. There we can see the two weighted layers and the skip connection. For the implementation we chose to have the same two layers with the shortcut to be Conv2d layers with a filter size of 3×3 and stride 2, as displayed in Figure 4.2.

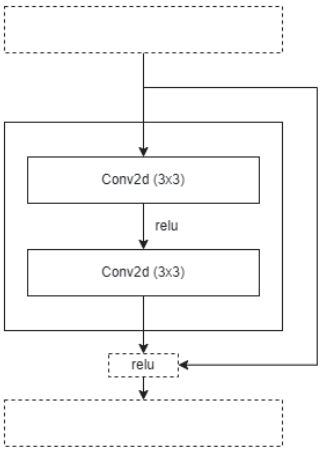


Figure 4.2: Architecture for the the custom residual Block

Our layers are now formed out of two of these residual blocks. For the sake of understanding and terminology we are going to call them Custom Residual Layers (CRL, 4.3). However, each of these CRLs has an initial Conv2d layer with kernel size 3×3 and stride 2, whereas the in features are some chosen value of x and the out features are double of x. This my sound confusing but will be clarified in just a moment.

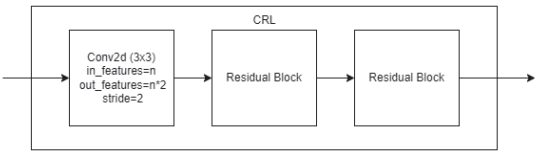


Figure 4.3: Architecture for the Custom Residual Layer

We chose to stack four of these CRLs onto each other, as seen in Figure 4.4. However, before the first CRL we put a simple Conv2d layer with in features 1 and out features as big as the resolution of the images, therefore 64 chosen to augment each sample from the initial $1 \times 64 \times 64$ dimensions to the new $64 \times 32 \times 32$ dimensions. Now that we do not have miss-matching layers, the first CRLs Conv2d layers only has stride one as well as the in and out features of 64 which results that the dimensionality is not reduced. Therefore, CRL-1 ends in $64 \times 32 \times 32$ which is forwarded to CRL-2. So, CRL-2 ends in $128 \times 16 \times 16$, CRL-3 in $256 \times 8 \times 8$ and CRL-4 in $512 \times 4 \times 4$. After the last layer, we decided to have a linear or fully connected layer reducing the $512 \times 4 \times 4$ dimensions to 48 which represents the dimensionality of our latent space. More details for the architecture of the residual auto-encoder can be found in the GitHub repository under `autoencoders/resnet_ae.py` [6]

[6] https://github.com/cptunderground/cvae-papyri/blob/main/autoencoders/resnet_ae.py

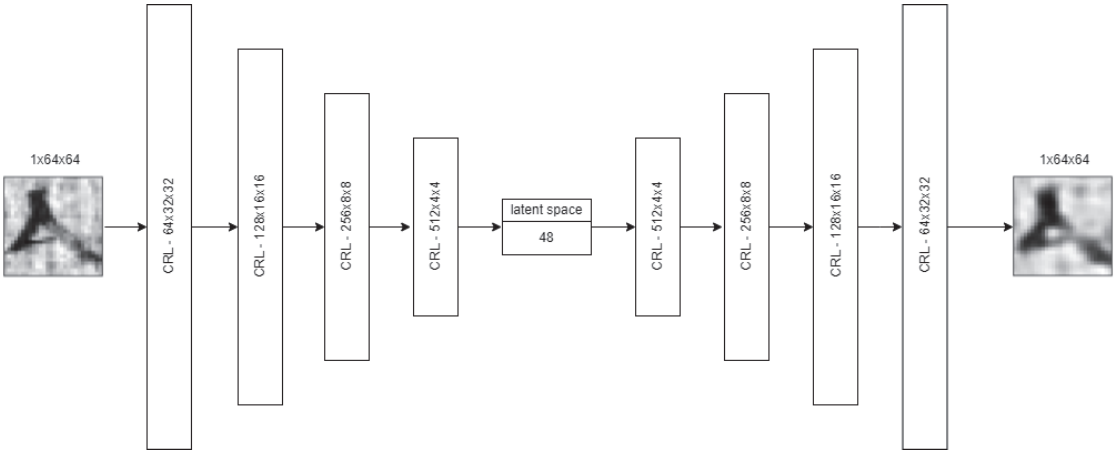


Figure 4.4: Full architecture for the ResNet Auto-Encoder

If we now want to choose a different input size, for example the resolution 128 by 128, the architecture only needs an additional CRL to match the previous dimensions. As a side note: Retrospectively, the last fully connected layer may be a too ambiguous choice, since the dimensionality reduction there is by the factor of 170. More on this in chapter 6 Future Work.

4.2.3 Training

Having the new pre-processed dataset and the model architecture, training the model is on the agenda. For this we need to define the optimizer and loss function. The obvious choice for this is to go with the state of the art, which we did. Chosen were the mean squared error loss (MSE-Loss) and the ADAM Optimizer.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2 \tag{4.1}$$

Figure 4.5: MSE-Loss Function

MSE is probably the most commonly used loss function in machine learning. It will never be negative, since every error is squared. This quadratic property also penalizes large errors more. Therefore MSE is sensitive towards outliers. On the other hand, if the model makes bad predictions, the errors get magnified. However, in praxis we do not care about the outliers. The aim is to have a good representation of the majority of the data, therefore the choice of MSE-Loss.

The ADAM [10] optimizer is our optimizer of choice. It is an optimization of the conventional gradient decent used to train neural networks. It has proven to be the state of the art

choice of optimizers by combining the advantages of two recently popular optimization methods as Kingma and Ba [10] state.

To decide when the model reaches the optimal state the data set gets split into a training, validation and testing set. We chose the ratio of 6:2:2 for this split resulting in 7113 samples for the training set and 2371 for validation and testing set. The model was trained with epochs ranging from 1000 to 10000 epochs. The optimal solution was always reached at epochs 200 to 300, when the validation loss reached the minimum. After these epochs, the training set is clearly over fitting by reducing the loss even more, whereas the validation and test loss is increasing again. As we can see in Figure 4.6, validation and test loss are more or less congruent. This is important and the behaviour expected. If the validation and test loss have different curves or diverge from each other, the data splitting is not representative, since the validation validates a different type of data than the test set tests in the end. Therefore, we can say the training results in an optimal model for this architecture that represents all of the available data.

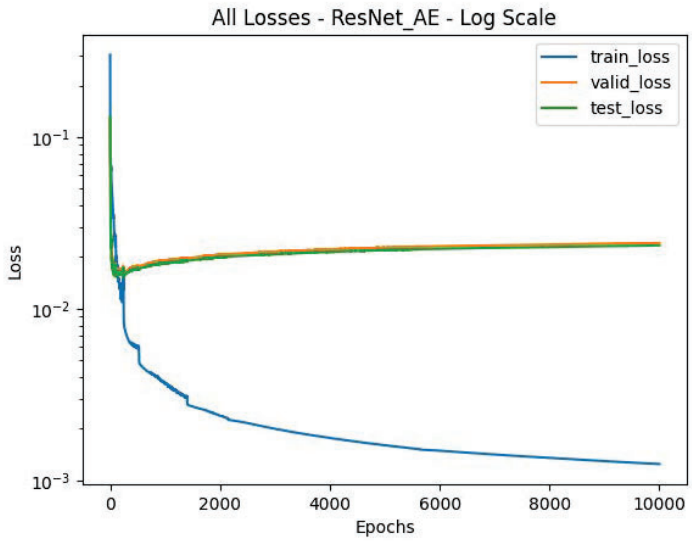


Figure 4.6: Training, Validation and Test Losses for the resnetAE

4.3 Evaluation

In Figures 3.8a, 3.10a, 3.8b and 3.10b we have seen the latent space for some encodings as four 7x7 pixel images where the values within the latent space vector are displayed as a pixel in grayscale. Therefore we can somehow see the full 196 dimensional latent space of one encoding. Drawing conclusions or even comparing encodings is rather hard with this abstraction. Fortunately, there are techniques that can provide remedy. In this section we introduce three techniques to analyze the latent space of our trained resnetAE. First, we dive into dimensionality reduction, second we

cluster the latent space and calculate the Rand index to compare the clustering with the ground truth and last but not least we analyze close samples in the latent space on similarity.

4.3.1 Dimensonality Reduction

Dimensionality reduction transforms data from a high-dimensional space into a low-dimensional representation. In addition, optimally dimensionality reduction does this in a manner, that the low-dimensional representation retains meaningful properties of the high-dimensional data. Arguably the best known algorithms for dimensionality reduction are T-distributed Stochastic Neighbor Embedding (t-SNE)[21] and Uniform Manifold Approximation and Projection (UMAP)[14].

t-SNE

t-SNE [21] is a nonlinear dimensionality reduction technique. t-SNE calculates the relation or better said, the probability, that two datapoints are related in the high-dimensional space. Afterwards it tries to produce a similar distribution by choosing low-dimensional embeddings. Therefore, t-SNE is not the most suitable algorithm to use for clustering or outlier detection, because it does not as a matter of course preserve densities or distances well.

UMAP

UMAP [14] is also nonlinear dimensionality reduction technique, very similar to t-SNE when looking at the results. UMAP assumes that all the data is uniformly distributed on a locally connected Riemannian manifold citeumap. In fact, as McInnes et al. [14] state: "The UMAP algorithm is competitive with t-SNE for visualization quality, and arguably preserves more of the global structure with superior run time performance. [...] making it viable as a general purpose dimension reduction technique for machine learning." Therefore, we stick with UMAP as our choice of dimensionality reduction tool.

Having this explanation about UMAP, it is time to look at an actual UMAP plot of our 48-dimensional space reduced to two dimension. In Figure 4.7 we can see the low-dimensional latent space representation of our full testset with all character labels. Clearly we can see three big clusters, upper left, upper right and lower middle. Also, within these big clusters, smaller clusters can be seen. Actually, it is still very hard to tell what exactly is in these clusters, since 24 labels are so many. In Figure 4.8 we skipped most of the samples of the testset and only visualize the representation of the characters alpha, epsilon, iota and omicron. These four characters are the most frequent ones in the dataset. Additionally, style-wise they should not over lap too much, except epsilon and omicron. These two classes can share the round "O" or "half-moon" feature. As we can see, we get a totally different representation in the lower dimensions. Here the clustering is telling more. Since we only have four characters and therefore

four colors, examination is much easier. It can be clearly seen, that the iotas are learned as an individual cluster, separated from the others. In contrast, alpha, epsilon and omicron are more or less evenly distributed in the big cluster on the right side of the plot.

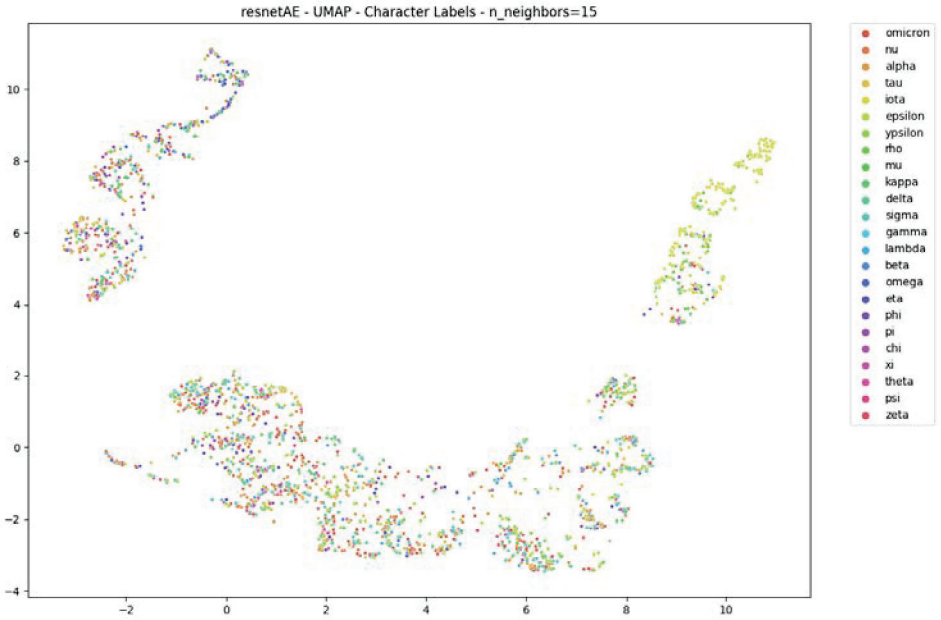


Figure 4.7: UMAP Plot of the full 48-dimensional Space with 15 Neighbors

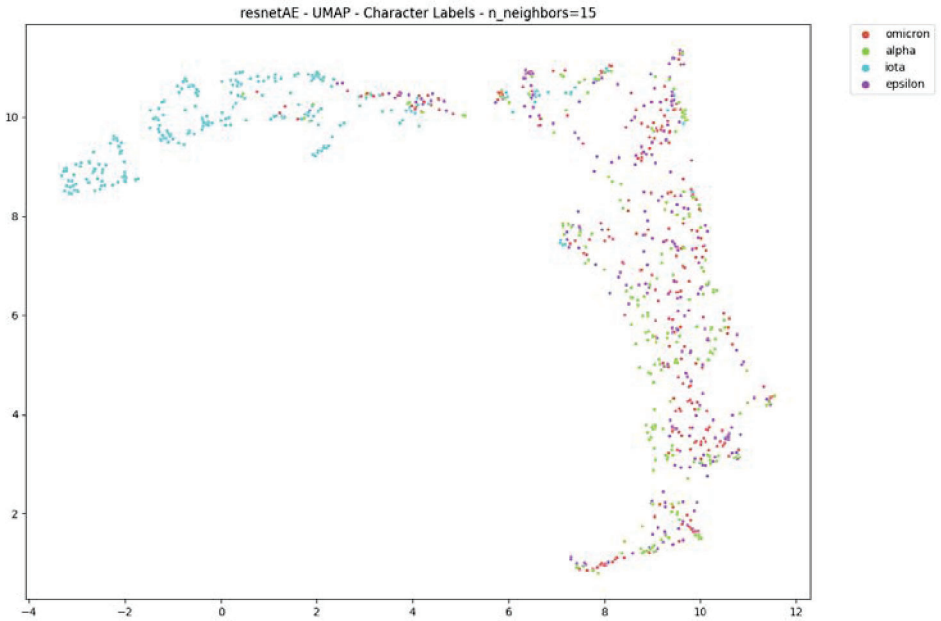


Figure 4.8: UMAP Plot of the 48-dimensional Space of Characters alpha, epsilon, iota and omicron with 15 Neighbors

Still, these plots can be optimized. The UMAP algorithm has the parameter `n_neighbors` to balance local versus global structure of the data. So, by adjusting `n_neighbors` parameter to a lower value, we force UMAP to concentrate on the local structure more. By increasing `n_neighbors` UMAP is pushed to look a larger neighborhoods and find a better description of the global structure. Therefore, this value shift either results in losing detailed structure for getting a big picture of the data or concentrates on the details to the detriment of the broad data.[7] In Figure 4.7 and 4.8 we used 15 as value for `n_neighbors`. This is a mid value compromising both ends of the spectrum. Now, interesting is what happens with a lower and higher value. In Figure 4.9 and 4.10 we can see the same thing as before, but with the adjusted `n_neighbors=5`. Figure 4.9 is still too overloaded to make any assumptions. However, Figure 4.10 is really interesting to look at. Now, since the lower value for `n_neighbors` we get more information about local structure. In the left, we can see within the big iota cluster even smaller clusters with probably similar iotas in them. Different adjustments of `n_neighbors=[2, 10, 20, 100, 200]`, both for all characters and the most frequent can be found in the Appendix under subsection A.2.1.

[7] According to: https://umap-learn.readthedocs.io/en/latest/parameters.html - latest version: 04.09.22

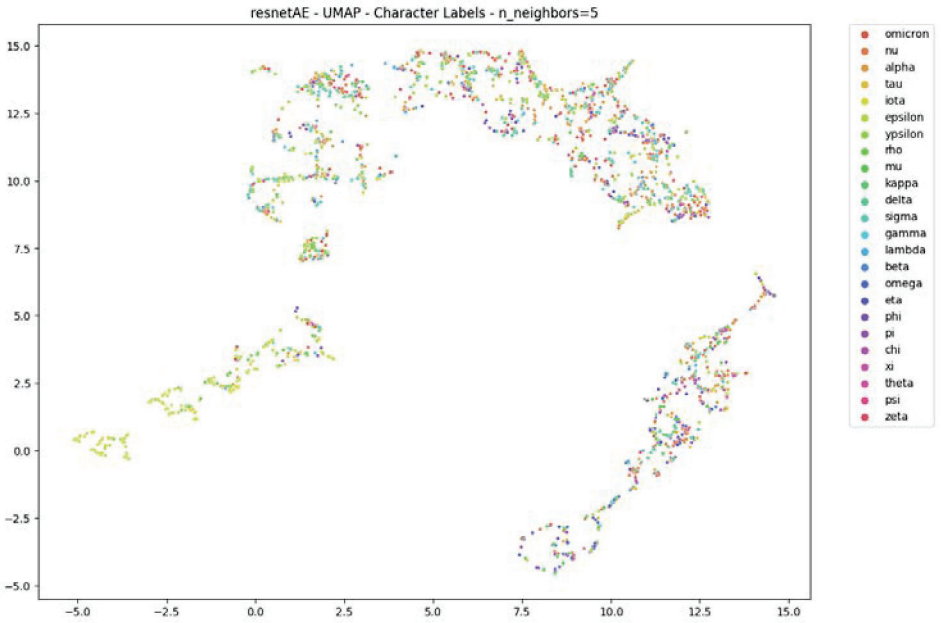


Figure 4.9: UMAP Plot of the full 48-dimensional Space with 5 Neighbors

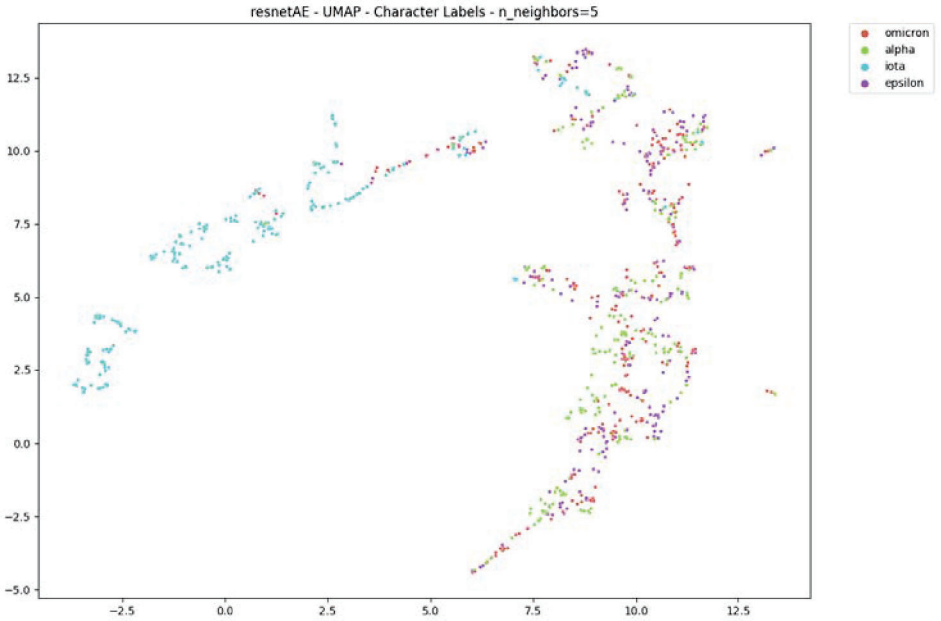


Figure 4.10: UMAP Plot of the 48-dimensional Space of Characters alpha, epsilon, iota and omicron with 5 Neighbors

However, in these plots we can only find information about the character labels and how they are distributed in the latent space. Actually, we can draw out more information about the latent

space, since we also have the fragment labels. Well, as we have sees, reducing the full latent space into a lower dimension and coloring all character labels is overwhelming for analysis. This is not going to get better with the 96 fragment labels. Therefore, we chose a similar approach to only have a look at the 6 most frequent ones. In Figure 4.11 we can see the same plot as in Figure 4.7 with the coloring of only the six most frequent fragment labels. We can see that no general clustering can be detected and therefore maybe the model does not learn the fragment labels without further help. However, we can see that the fragments for some sort of bands throughout the plot. This on the other hand, can be an indication that some fragment features are learned. Again, we can have also a look at the more locally detailed plot of these fragments. In Figure 4.12 we can see the plot for `n_neighbors=5`. The behaviour and formed bands are similar. Therefore, we cannot draw any more information about the learning of the fragment labels. Aigain, different adjustments of `n_neighbors=[2, 10, 20, 100, 200]`, for the most frequent fragments can be found in the Appendix under subsection A.2.1.

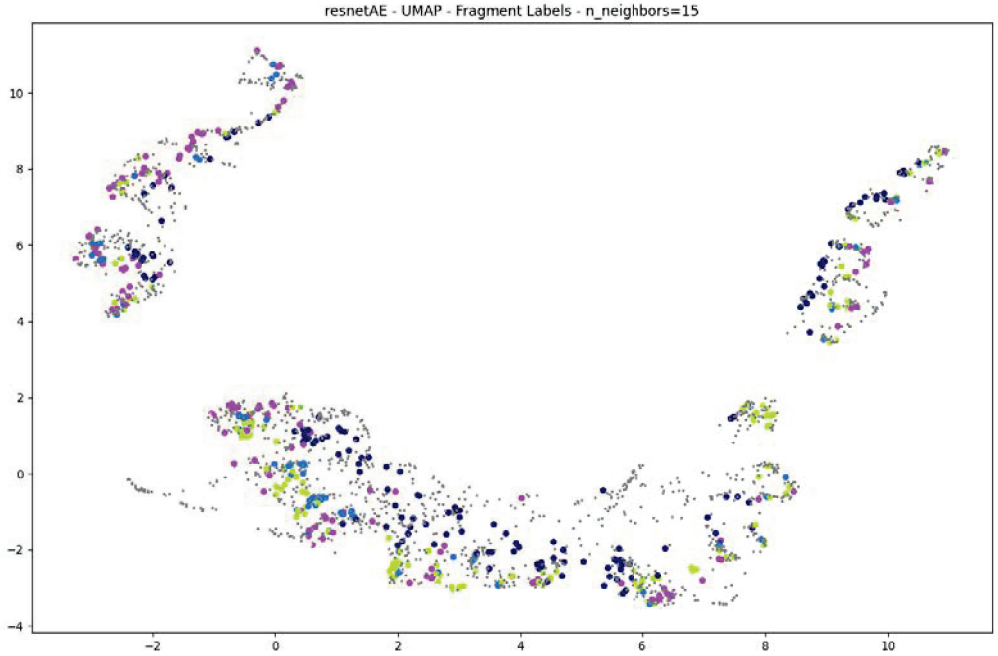


Figure 4.11: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 15 Neighbors

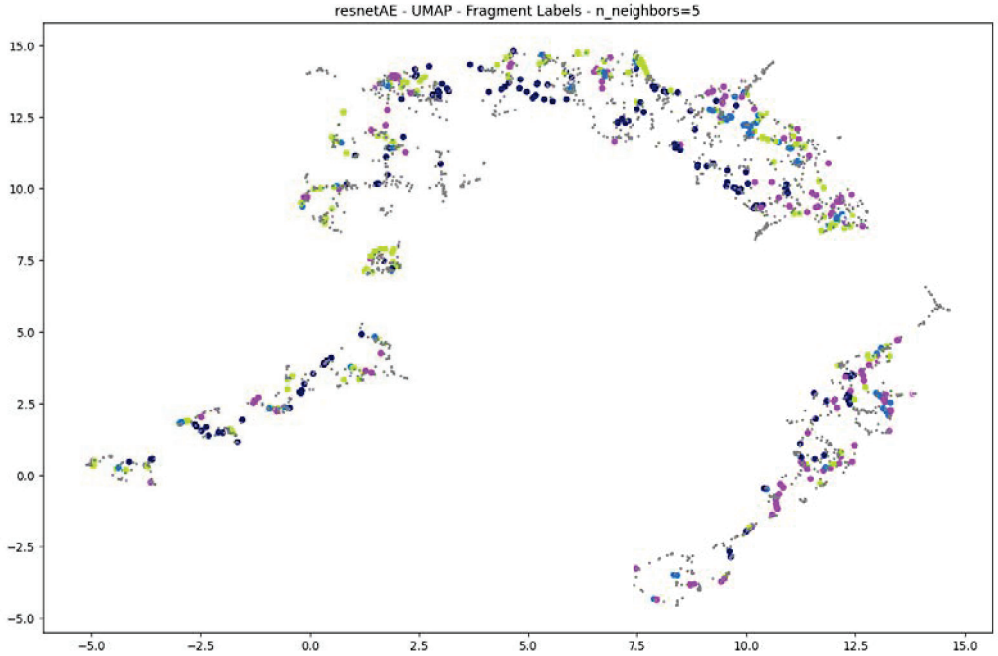


Figure 4.12: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 5 Neighbors

4.3.2 K-Means Clustering and Rand Index

In the previous section we have seen UMAP plots with different parameters and colored by different labeling. We assumed to see big clustering for iotas and smaller clustering for several other labels. This needs to be verified. In this section we have a look at the real clustering of the latent space and also how we can draw measurements.

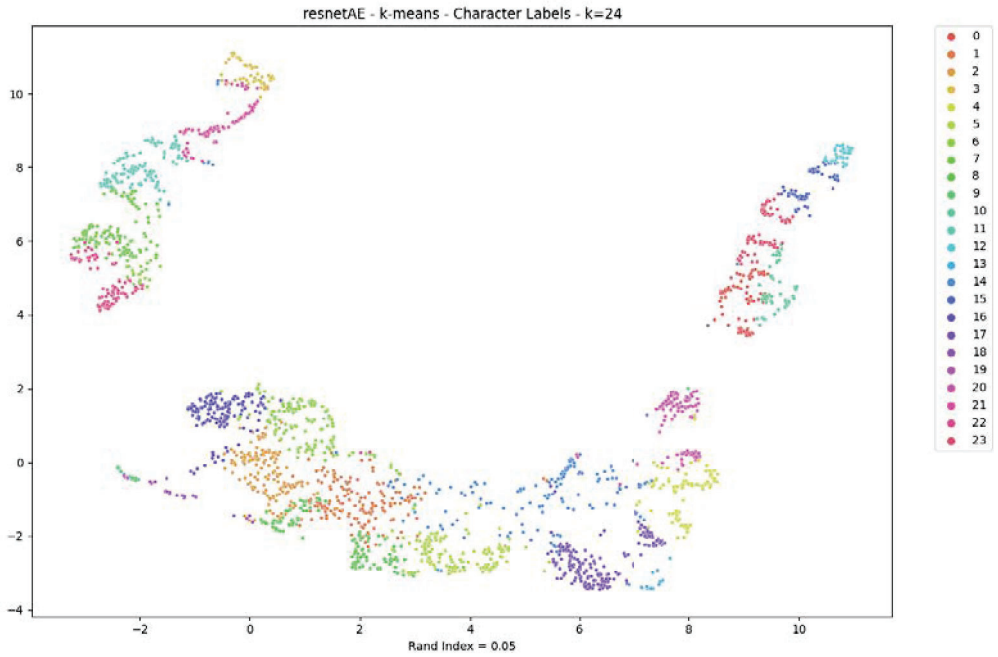


Figure 4.13: K-Means with 24 Clusters on Full Latent Space

In Figure 4.13 we can see how the k-means algorithm would cluster the latent space into 24 different clusters, reduced to 2 dimensions. Clearly, these clusters are very well distinguishable from each other. This is important. The k-mean clustering takes place in the 48-dimensional latent space and assigns a label to each datapoint. UMAP now reduces the dimensionality and the lower representation gets colored by the newly assigned labels. Therefore, when looking at Figure 4.13 we can see that dimensionality reduction preserves the latent clustering. However, we do not know if the clustering exactly matches the original embedding of the ground truth in the full latent space. To actually measure the similarity of the clustering, we introduce the technique of the Rand index. As we can see in Figure 4.13 the Rand index scores a value of $R = 0.06$ or 6%. But, what is the Rand index exactly?

Rand Index

The Rand index or Rand measure is a measurement that compares two clusterings on similarity. R the Rand index is calculated as follows: $R = \frac{a+b}{n(n-1)/2}$ where:

- a is the number of times an element pair belongs to the same cluster for both clustering methods.
- b is the number of times an element pair belongs to different clusters for both clustering methods.
- $n(n-1)/2$ is the number of unordered pairs in the set of the n elements.

So the Rand index R always takes on a value between 0 and 1. Whereas 0 means that the two clustering methods do not agree on any pair of elements in the clustering. In contrary, a Rand index value of 1 denotes that the two clustering methods perfectly agree on every pair of elements in the clustering. This is still a little hard to grasp, so the example below shows the complete calculation of R:

Suppose we have a dataset of 4 elements: $\{A, B, C, D\}$.
Further, the two clustering methods cluster the dataset as follows:

- Method 1: $\{1, 1, 1, 2\}$
- Method 2: $\{1, 1, 2, 2\}$

To now calculate the Rand index we need to first identify all $n(n-1)/2$ where $n = 4$, therefore 6 unordered pairs between the data points.

$\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}$

As the next step, the calculation of a is needed. Again, a represents represents the number of unordered pairs that belong to the same cluster for both methods. Only the pair of $\{A, B\}$ is in both methods assigned to the same cluster, hence $a = 1$. Now, we calculate b in the same manner. The only difference is we now search for every pair that is not assigned to the same cluster for both methods. Here, $\{A, D\}, \{B, D\}$ are the unordered pairs that are assigned to different clusters. So, $b = 2$. We now can calculate the Rand index for these two clusterings:

$$R = \frac{a+b}{n(n-1)/2} = \frac{1+2}{4(4-1)/2} = \frac{3}{6} = \frac{1}{2} = 50\% \tag{4.2}$$

This means the two clustering methods agree on half the data to be clustered in the same manner for both.

Therefore, the Rand index of 5% for our clustering is not very promising. Also, with different parameters for the numbers of clusters, the k-mean clustering only reaches from 3% to 8% accuracy on the Rand index. However, one promising clustering plot emerged out of the evaluation. In Figure 5.7 we can see the only the latent space clustering of alpha, epsilon, iota and omicron, clustered by k-means with four clusters. As we can see, the Rand index scores a similarity of $R = 0.188$ which is the best score acquired during all of the evaluation. Concluding, seeing the best results by only analysing the letters alpha, iota, epsilon and omicron may be a hint to approach the dataset differently.

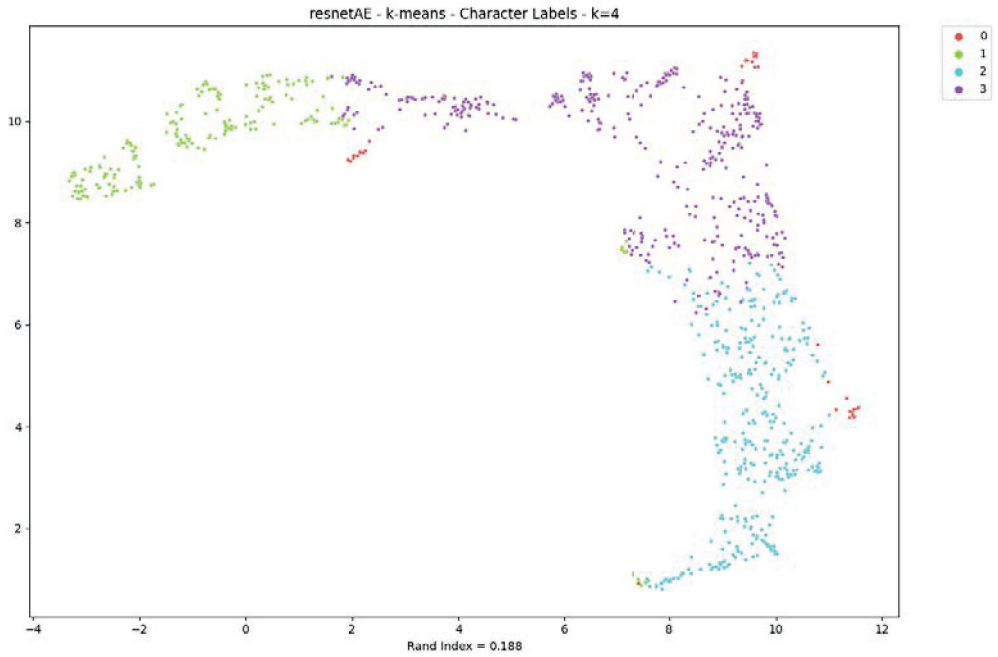


Figure 4.14: K-Means with 4 Clusters on the Latent Space of alpha, iota, epsilon and omicorn

So this means the clustering algorithm can not cluster the latent space in a manner that the clustered labels match the real labels of the data. Therefore, a different measurement is needed to interpret the latent space and tell how good of a clustering we have present.

4.3.3 Euclidean Distance on Latent Space

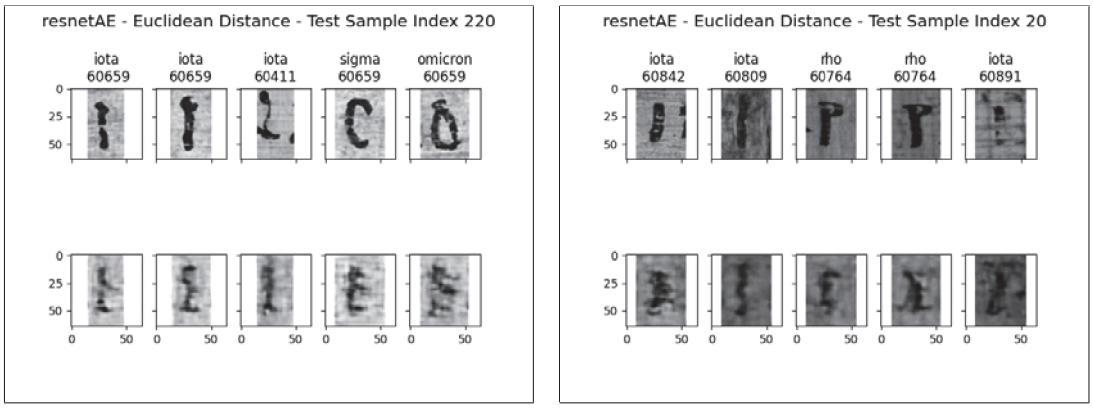
Since the Rand index has not lead us to a better conclusion on the interpretation of the latent space, a we introduce a new way to analyze the latent space. With the help of the euclidean distance we calculate the closest samples to a base sample in the latent space. Let us discuss this in more detail.

Having sample the encoded sample i and therefore its 48 dimensional latent space vector. We now calculate the euclidean distance to every other sample encoded to its latent space vector of the testing set including the base sample and sort them from lowest to highest. Now we have a list where the first entry is the base sample with a euclidean distance of 0 to itself whereas the second entry has the lowest distance to the base sample and is therefore closest in the latent space and so on for all other samples.

$$d(i,j) = \sqrt{(i_1 - j_1)^2 + ... + (i_{48} - j_{48})^2} \tag{4.3}$$

Figure 4.15: Euclidean Distance for the base sample i and some testing sample j

We can now have a look at the samples that are close to each other in the latent space. Aligning the original cliplets and the decoded cliplets with their character and fragment labels in a picture gives us visual and informational feedback that we can interpret. First and foremost what are we looking for. Since we are analysing style the desired outcome of these lists would be either that we see the same character labels near each other so the model learns the difference of style via the actual letter or that we have the same fragment labels close to each other, therefore the model learns style from the actual difference in how the letters are written in the same or different fragment. On the other hand, a mixture of both is also desired as stated in the hypothesis. Before the analysis it needs to be noted, that the test set contains of 2371 samples. Since this analysis is based on anecdotal evidence verified by eye, a sample size of 2371 is too large. Therefore we only analyzed every tenth sample of the test resulting in 237 base samples. Terminology-wise, if we talk about for example sample 220 this actually means the sample at index 220 of the test set.
In Figure 4.16a we can see such a desired outcome of the euclidean distance. In the upper left corner we can see our base sample 220. Next to this iota there are two more iotas which are the closest to the base sample in the latent space. Further, we can see a sigma and an omicron with the same fragment label of 60659. In conclusion, the model learned a compound representation of the character labels and the fragment labels. In Figure 4.16b we have a similar behaviour. The base sample 20 has also iotas as closest samples in the latent space. Yet, there are two rho in there. We can see in the reconstruction below that in all 5 samples the vertical stroke is the most present feature and this could be why these two rho are that close in the latent space to the base sample 20.



(a) Sample 220 (b) Sample 20

Figure 4.16: Sample 220 & 20

An even better sample concerning character labels we can see see in Figure 4.17. There, we can see five iotas in a row. The similarity can already be seen by eye in the upper row where we see the originals. A more significant role plays the lower row, where we can see the decoded

images from the latent space. As there is even more similarity, the model clearly learns the inputs features and can reconstruct them. Actually, this figure is fully represents the hypothesis. We have a single character class label iota and five different fragment labels. Therefore, we can assume that the iotas of these five fragment labels are written in the same style and therefore by the same author.

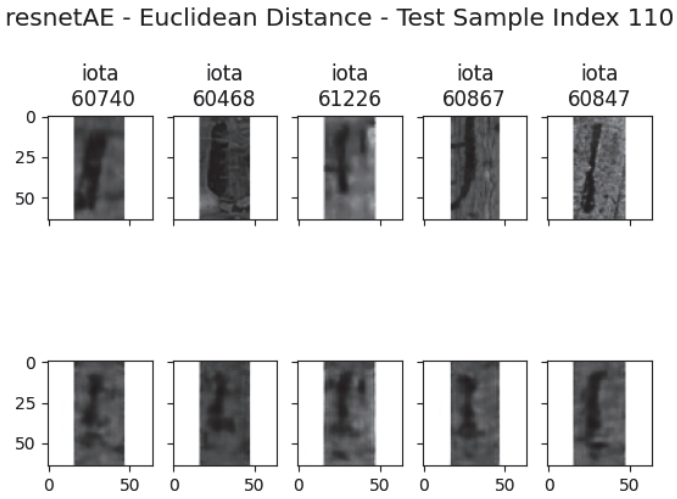
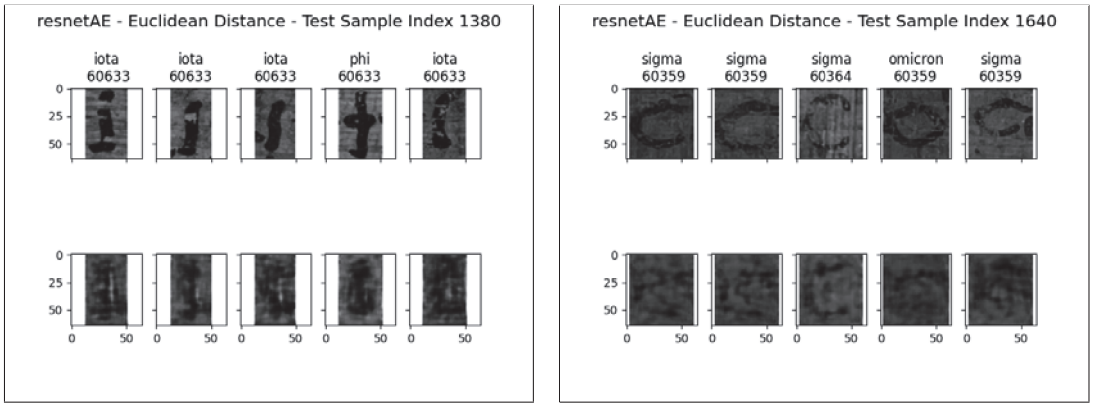


Figure 4.17: Euclidean Distances of Sample 110

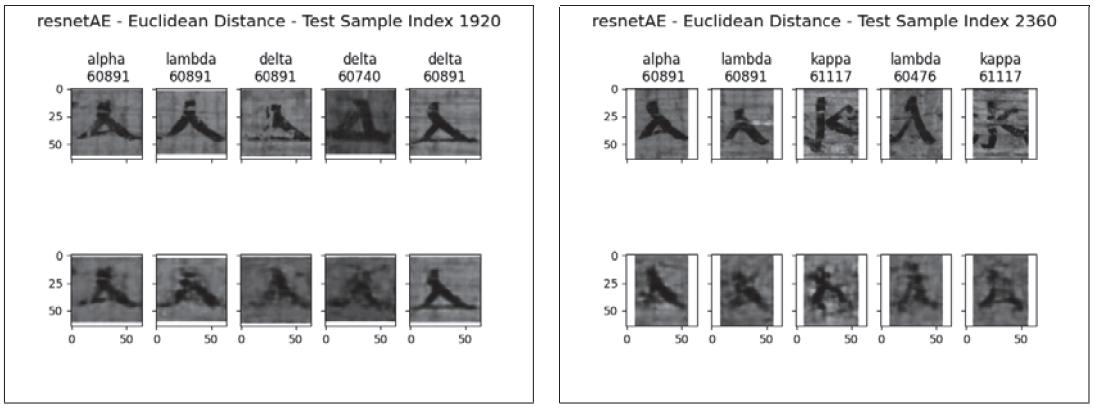
Further, sample 1380 and sample 1640 in Figure 4.18 shows a good example, that not only character labels get learned but also fragment labels, that we connect to style. Again, the hypothesis is, characters that originate from the same fragment should be written by the same writer. Therefore, the characters should be written in the same or similar style. Now, examining Figure 4.18a we can see that all fragment labels are the same and therefore, the cliplets written in the same style. As we also can see, there is a phi instead of an expected iota. On the other side in Figure 4.18b we have a sigma as base sample, the third and fourth sample in the Figure are outliers. The third sample has not the right fragment label and the fourth sample is an omicron instead of a sigma. Yet again, comparing sigmas and omicrons is already a hard task since the share the round ”O” and ”C” feature that can be confusing and very close to each other due to handwriting style. But, this is exactly what we want to analyze. With these two samples we can surely say that iota and phi as well as sigma and omicron share the same or similar features. Also, again we can assume based on 4.18b that maybe fragment 60364 and 30359 are written by the same author.



(a) Sample 1380 (b) Sample 1640

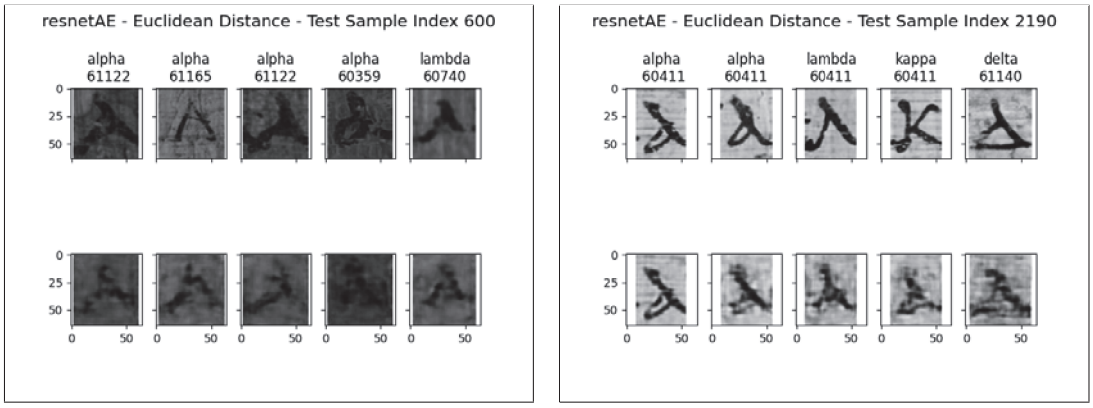
Figure 4.18: Sample 1380 & 1640

A further example that shows the same behaviour can be seen in Figure 4.19 and Figure 4.20. In Figure 4.19 we have two different alphas from the same base fragment 60891. As we can see the in both sub plots, the model confuses alphas with lambdas and deltas, as well as some kappas. Again, by eye we can see that the alphas, deltas and lambdas all share the triangular or hat base feature. We can verify this quickly by just looking at the reconstruction where we can clearly see the decoded images forming full triangles or partial triangles (hats). For the kappas it is a little bit odd, the hat feature is only present in the lower half of the kappa. In the reconstruction of the kappas we can see that the upper half is not learned or just poorly learned. This can be a indication that the latent space has not enough dimensionality to hold onto all information. Further, as we have already seen in the UMAP plots, maybe the dataset holds too much different information. Instead of training the model with all labels, a subset of only the four most frequent labels is the better approach. But more on this in chapter 6 Future Work.



(a) Sample 1920 (b) Sample 2360

Figure 4.19: Sample 1920 & 2360



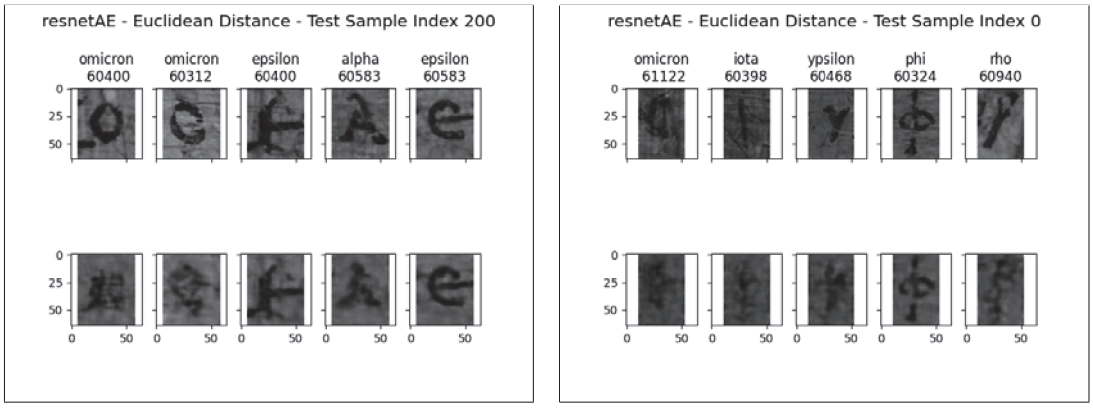
(a) Sample 600 (b) Sample 2190

Figure 4.20: Sample 600 & 2190

We can see this behaviour at many other samples. Figures for this can be found in the Appendix under subsection A.2.3. In the following list we can find shared features throughout the dataset and the class labels:

- alpha A, lambda Λ, delta Δ and kappa κ share the triangle feature (see 4.19, 4.20 4.17)
- tau τ and ypsilon Υ share the T feature (see A.26, A.21)
- sigma σ, epsilon ϵ and omicron o share the O and C feature (see 4.18, A.23)
- iota ι, roh ρ and phi ϕ shares the vertical line or I feature (see 4.16, 4.18)
- nu ν, mu μ and eta η share the N feature (see A.24)
- eta η, nu ν and pi π share the H feature (see A.22)

Therefore, the model is capable of representing most of the data and also learns shared features. However, we want these shared features to be disentangled and our model is not capable of doing this. It is a result of the unsupervised learning. Our model has no information about the labels yet. The loss is only calculated between the input and the reconstruction. Obviously, for example alphas, deltas and lambdas are close to each other in the latent space because of the shared triangular shape. We can improve this by adjusting layers, latent space size, loss function, learning rate and so on. However, since we have labels, supervising the learning is also an option. Introducing a conditioning on the labels will add a bias to the loss calculation and the model has more information to distinguish alphas, deltas and lambdas from each other.



(a) Sample 200 (b) Sample 0

Figure 4.21: Sample 200 & 0

Apart from all these good examples, where we either see obvious learning or can interpret some the distances, there are samples where the behaviour of the model is random or hard to explain. In the example Figure 4.21a we see a on a first sight bad "clustering". When having a closer look, we still see two omicrons and the alpha and epsilon with the same fragment label 60584. We are not sure why these samples have the closest distance to the base sample. Still we can assume that the model maybe learns somehow the background as well and therefore these samples are this close to each other. Having a look at Figure 4.21b We can see no correspondence of either character labels or fragment labels. Fortunately, the non-interpretable samples do not over weigh the ones that have explainable distances. A big problem for this explainability is that we do not really know much about the fragment labels. Unfortunately, there is no knowledge about the ground truth distances from one fragment to another in the dataset as we use it now. For example, in Figure 4.21b we have the fragment labels 61122, 60398, 60468, 60324, 60940. We have no information where these fragments come from inside the Iliad. If these five fragments actually are close to each other within the original Iliad, then the interpretation of this sample is completely different and switches from a "bad" one to a "good" one. Still, we are lacking this information and can only provide a guess instead of profound explanation.

5

Two Step Training with Conditional Variational Auto-Encoder

In the previous chapter we have seen that the auto-encoder based on the ResNet[9] architecture has already provided promising results. Now, the architecture can be fine tuned. Instead of altering the ResNet architecture, by for example adding layers or adjusting the latent space dimension, we explore a different approach to see if the results can be improved. The idea is to have two step training approach. More precise, the latent space of the resnetAE shall be conditioned on the character labels as well as the fragment labels with the help of a Conditional Variational Auto-Encoder (CVAE). This means, as soon as the optimal model is trained for the resnetAE, two CVAEs condition the latent space on either of the labels. This idea came from further investigation on the usage and augmentation of auto-encoders. Therefore, this is no novel idea, similar approaches already have been made[8][6][16]. However, first a theoretical introduction to Variational Auto-Encoders and Conditional Variational Auto-Encoder is given. Afterwards, we show the implementation of such a CVAE and the architecture chosen for this approach. Last, we evaluate our results in the same manner as the resnetAE to see if the results improve or not.

5.1 Methodology

5.1.1 Variational Auto-Encoder (VAE)

The Variational Auto-Encoder [11][7][17] is an expanded form of the original auto-encoder architecture. The latent space is not a fixed vector anymore, but a composition of a mixture of distributions. By paving the space with distributions, the VAE has a denser representation of the latent space. The VAE learns more than just a single point. This is done by maximizing the log likelihood of our data $P(X)$ under some "encoding" error. The original VAE model has the two parts: the encoder $Q(z|X)$ and the encoder $P(X|z)$.

So the objective[8] is:

$$logP(X) - D_{KL}[Q(z|X)||P(z|X)] = E[logP(X|z)] - D_{KL}[Q(z|X)||P(z)]$$

5.1.2 Conditional Variational Auto-Encoder (CVAE)

VAE [11][7] encoders model the latent space variable z directly based on the given data X. The whole dataset, all letters we feed into the VAE in our case. So the VAE does not care about any labels. The same applies to the decoder part, it only models X directly based on the latent variable z without any information about any other properties such as the labels. A VAE can be improved to a CVAE [22] by conditioning the encoder and decoder on another property. This other property is c, so the encoder is now conditional to two variables X and $c : Q(z|X, c)$. The same with the decoder, it's now conditioned to two variables z and $c : P(X|z, c)$.
This means that the objective[9] changes to:

$$logP(X|c) - D_{KL}[Q(z|X, c)||P(z|X, c)] = E[logP(X|z, c)] - D_{KL}[Q(z|X, c)||P(z|c)]$$

We just conditioned all of the distributions in the latent space with a variable c, for example the label. Therefore, the real latent variable is distributed under $P(z|c)$ resulting in a conditional probability distribution. This means that for each c, there exists a different $P(z)$. The encoded latent space is now a composition of a mixture of distributions for every c possible.

Therefore, with a CVAE we transfer the unsupervised learning of an auto-encoder on a fixed latent vector to a supervised training with a latent distribution. The unsupervised training and validation by calculating the loss between the input and the decoded output is still part of the CVAE. Yet in addition, also the labels for the input are taken into account and condition the encoding on these given labels.

5.2 Implementation

Having the background on VAEs and CVAEs the implementation might seem tricky. Actually, with some little tricks it is really manageable. By adding a one-hot encoding to the inputs, the labels get also reconstructed and validated by the loss-function.

5.2.1 Loss Calculation of Conditioned Input

In theory, one might see a rather big difference between a VAE and a CVAE. In practice VAEs and CVAEs are very close. As we have seen the objective of the VAE in subsection 5.1.1 in

[8] Equation based on [11]
[9] Equation based on [22]

theory, in practice this is calculated as a loss function.

$$Loss_{VAE} = E[logP(X|z)] - D_{KL}[Q(z|X)||P(z)] \tag{5.1}$$

Where E is the direct reconstruction loss. This term compares the model output with the model input and returns the generative loss. The loss function however can be chosen freely. The second term, the Kullback-Leiber divergence compares the latent vector and penalizes the VAE on choosing latent vectors not on the desired distribution. For the CVAE the objective changes with the conditioning on some label c.

$$Loss_{CVAE} = E[logP(X|z,c)] - D_{KL}[Q(z|X,c)||P(z|c)] \tag{5.2}$$

Mathematically this seems hard. However, in practice there is a trick to it. By encoding a one-hot vector with the label of a sample and concatenating this vector to the actual input for the auto-encoder, a new input vector is generated. Now the CVAE encodes and decodes the data as well as the labeling and therefore the labeling or conditioning is taken into account by the reconstruction loss as well as the latent loss.

5.2.2 Architecture

The architecture of the two CVAEs are similar and rather simple. As said, the input layers dimensionality is the addition of the latent space vector of the resnetAE and the number of labels as a one-hot encoding. This results for the character label conditioning (charCVAE) in $48 + 24 = 72$ input dimensions and for the fragment label conditioning (fragCVAE) in $48 + 96 = 144$ input dimensions. The inputs are fed through two fully connected layers in the encoder, whereas the first layer has the same output size as the input and the second layer has the output size of 48 to match the dimensionality of the resnetAE latent space but conditioned on either the fragment or character label. The decoder is as expected just the inverse of the encoder. Figure 5.1 shows the full architecture of the combined resnetAE architecture and CVAE architecture. More details for the architecture of the CVAEs can be found in the GitHub repository under `autoencoders/char_CVAE.py` [10] and `autoencoders/frag_CVAE.py` [11]

[10] https://github.com/cptunderground/cvae-papyri/blob/main/autoencoders/char_CVAE.py
[11] https://github.com/cptunderground/cvae-papyri/blob/main/autoencoders/frag_CVAE.py

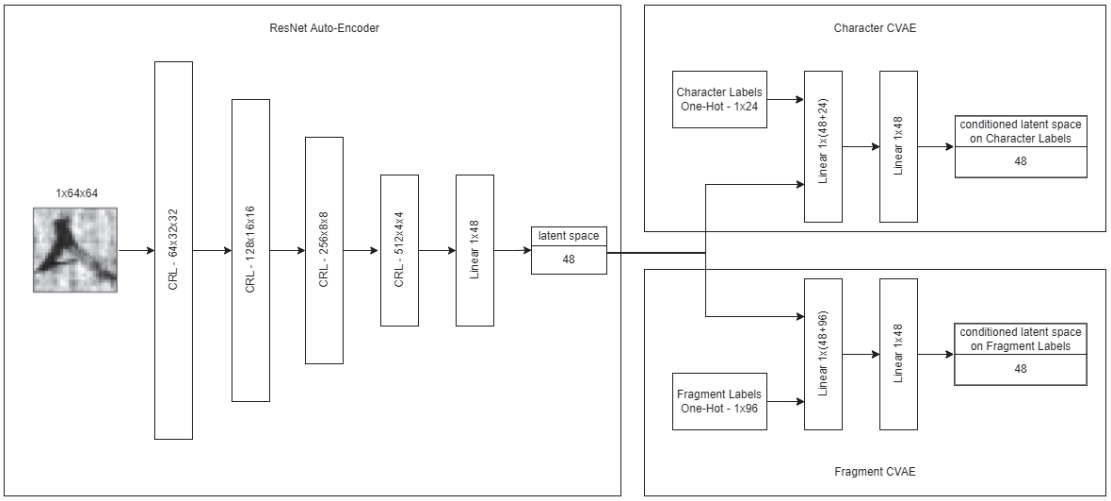
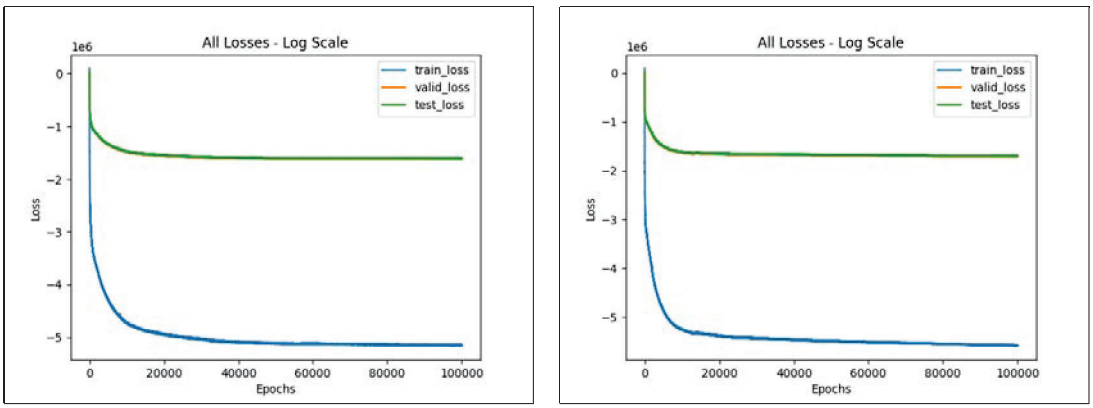


Figure 5.1: Architecture for the Two Step CVAE Approach.

5.3 Evaluation

As we already have a evaluation pipeline shown in section 4.3, we can draw the same measurements for the two step CVAE architecture. To already anticipate the results, the evaluation of both conditioning on the character labels as well as the fragment labels has not improved the style analysis in particular. However, already the training shows unexpected behaviour. Even after 100'000 epochs of training no optimal model has been found and the validation loss is still decreasing (5.2). This is very atypical. But first, let us have a look at the actual results.



(a) charCVAE (b) fragCVAE

Figure 5.2: Training Losses for charCVAE and fragCVAE

5.3.1 Dimensionality Reduction

Back in the previous Chapter about the ResNet auto-encoder we have seen, that dimensionality reduction is an essential tool to analyze the latent space of our encoding. Again, in this Chapter we will also have a look at the full and partial latent space as before and compare what has changed and if we can draw new conclusions.

Starting with Figure 5.3 we have the full latent space conditioned on the character labels. Again, all the labels colored in one plot can be too much information. However, we can see that the structure of the low-dimensional representation has changed drastically. We do not see any big or small clusters by eye. Now, we rather have some sort of homogeneous distribution of the data. In contrast to the sketchy training, this is a good sign. The latent space therefore is conditioned and the encoded samples lay on the desired distribution proposed by the CVAE and verified by the Kullback-Leiber divergence. Once again, by only looking at our four most frequent character labels or changing the `n_neighbors` parameter, we can still maybe find clusters or draw new conclusions. Unfortunately, further evaluation has not lead to any better results. All results can be found in the Appendix under A.3.1. Still, we want to show the plot of the most frequent character labels with `n_neighbors=5` in Figure 5.4. Once more, the representation has changed to a more homogeneous distribution. However, we can not see clear clusters. The only thing that could be said is that on the left side we find more isolated iotas than on the rest of the representation. Still, this is a very vague assumption. Again, we will verify these "found-by-eye" clustering with the Rand Index later on.

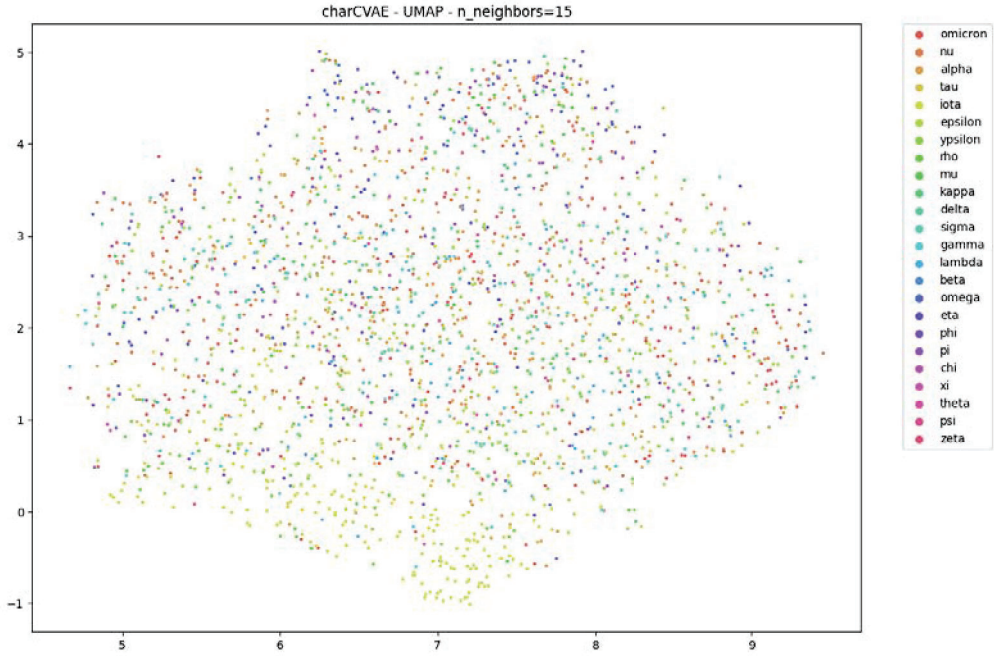


Figure 5.3: UMAP Plot of the full 48-dimensional Space with 15 Neighbors

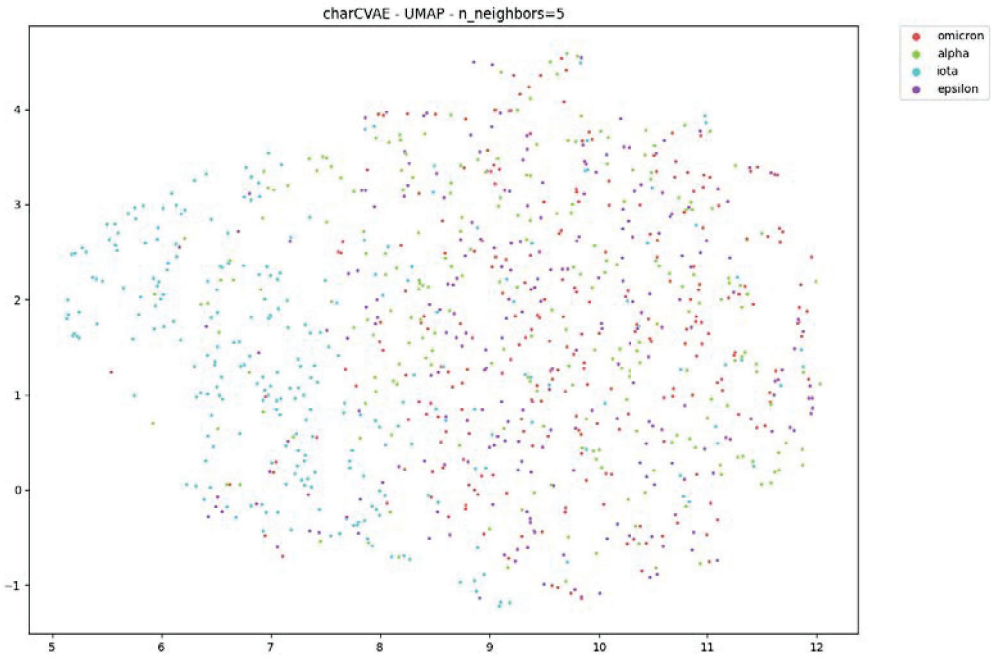


Figure 5.4: UMAP Plot of the Latent Space of the most frequent Character Labels with 5 Neighbors

Nevertheless, we can also analyze how the conditioning on the fragment labels affected the latent space and its lower-dimension representation. In Figure 5.5 and 5.6 we can see these plots, once with `n_neighbors=15` and once with `n_neighbors=5`. Unfortunately again, here, the even distribution of the fragment labels is even better to be seen. Even with changing up the `n_neighbors` parameter to different values, the distribution seems not to be changed as we can see in the further plots in the Appendix under A.4.3. Therefore, on first sight, the conditioning did not behave as expected. Still, we have more measurements to compare the two step approach with our ResNet base model.

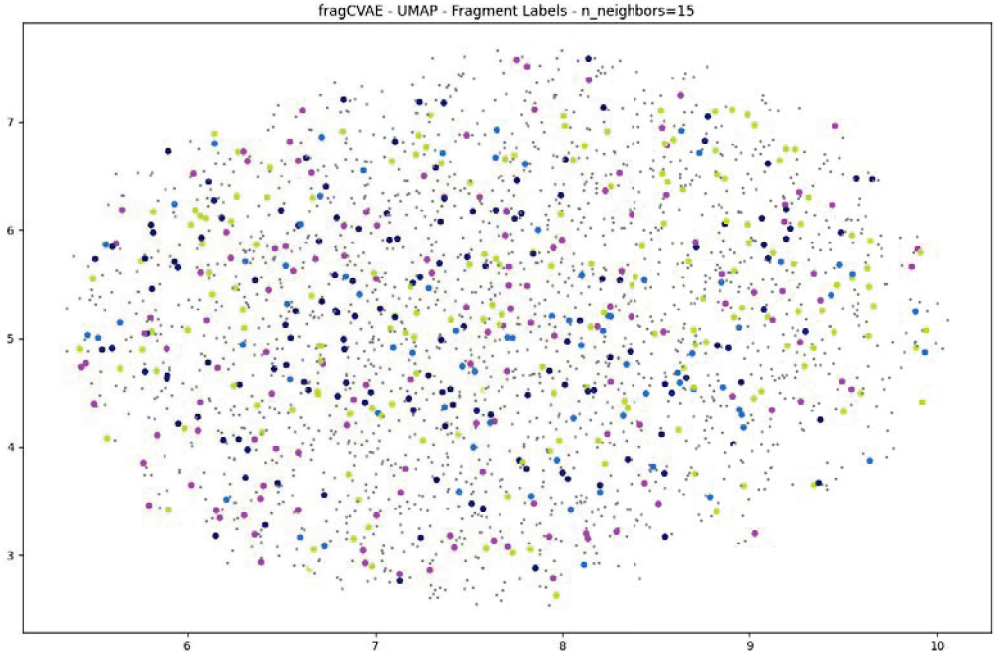


Figure 5.5: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 15 Neighbors

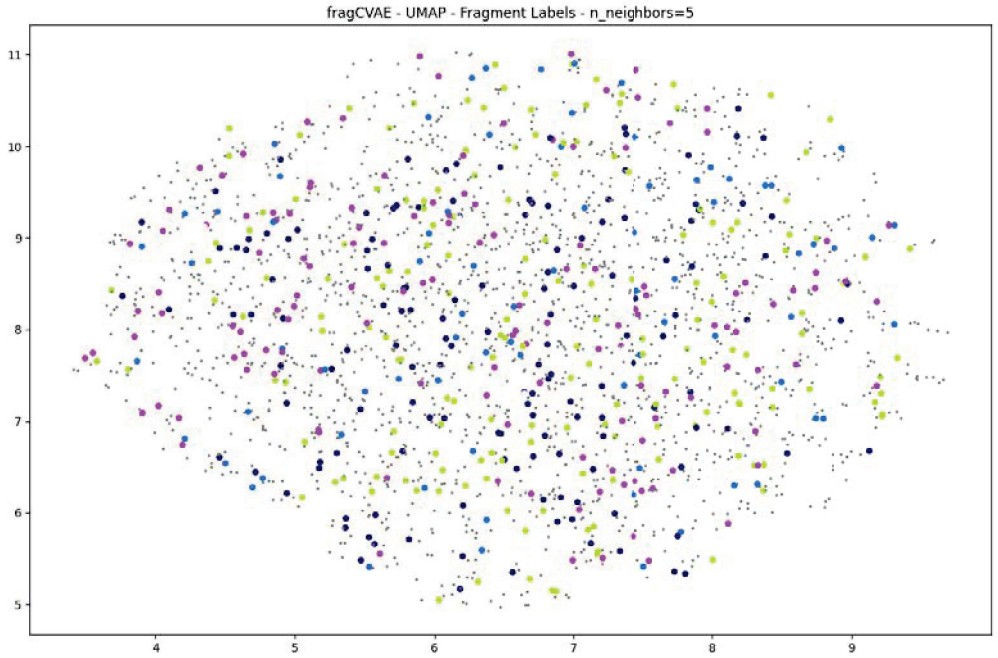


Figure 5.6: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 5 Neighbors

5.3.2 Clustering and Rand Index

Again, the two step CVAE architecture went through the same evaluation steps as the resnetAE discussed before. Therefore, we have also calculated the Rand index between the k-means clustering and the ground truth for the charCVAE. Again, the calculated Rand indexes are humbling. The agreement on clustering is even worse than before. For any clustering configuration (A.3.2), the rand index never exceeded 14%, whereas most of the configurations did not even break the 8% mark at all. Again, the best Rand index scored is for our four most frequent character labels clustered by k-means with 4 clusters reaching $R = 0.137$. This is worse than before, whereas our hypothesis would expect higher an therefore better scores. However, the assumption on splitting the dataset into a smaller subset and initially only train and analyze tor example all alphas, epsilons, iotas and omicrons is reinforced.

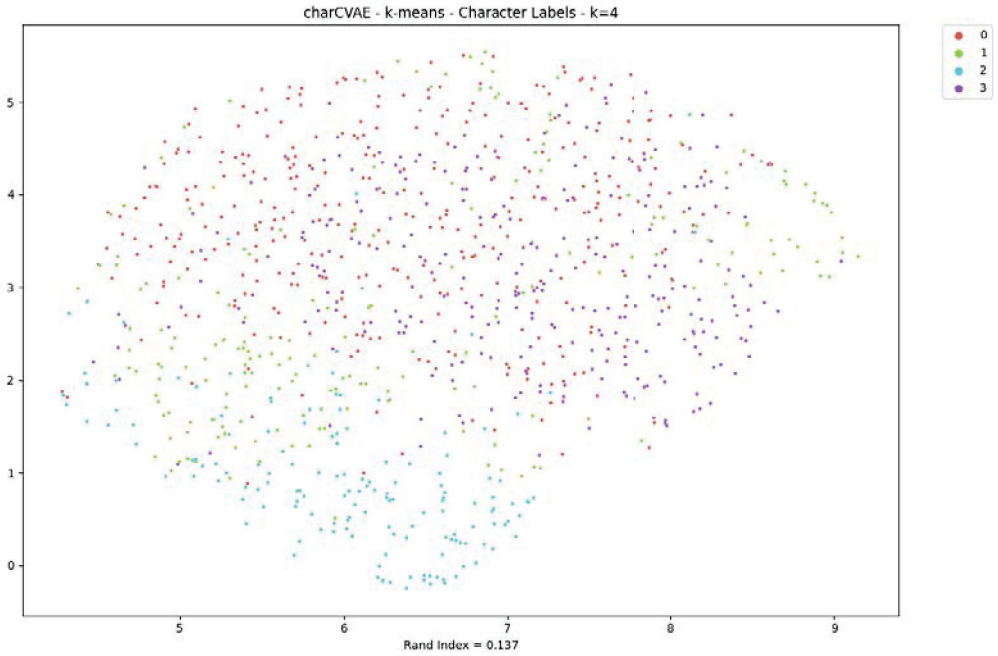


Figure 5.7: K-Means with 4 Clusters on the Latent Space of alpha, iota, epsilon and omicron

As we have seen in the residual architecture's evaluation before, this does not have to be bad in particular. The analysis with the Euclidean distance in the latent space has lead to a better understanding of the latent space.

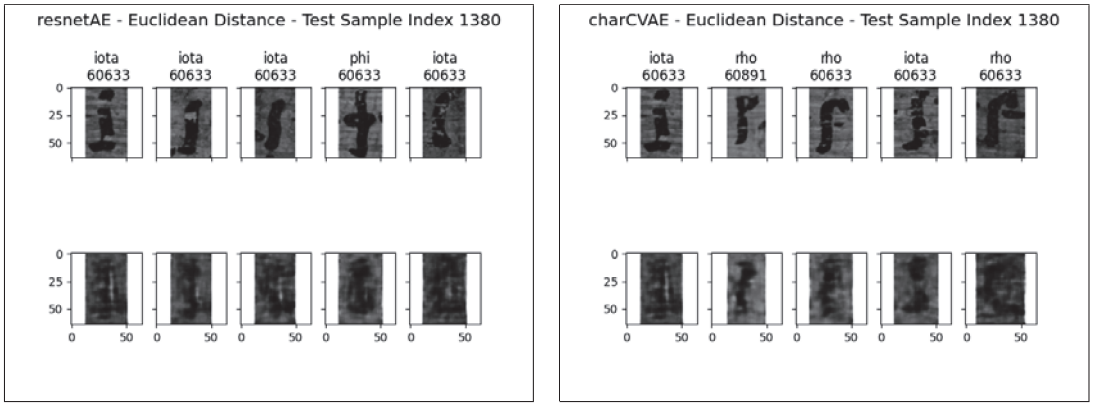
5.3.3 Euclidean Distance

Analysing the Euclidean distance between samples in the latent space has proven to be a useful and meaningful evaluation tool to understand how the latent space really looks like. Again, in this section we want to analyze the Euclidean distances between samples that went trough both

of our two step trained CVAEs. First, we have a look at the charCVAE that conditions the latent space on the 24 character labels and see what changes to the distance to other samples from our base samples selected in the resnetAE. Second, we evaluate the fragCVAE that conditions the latent space on the 96 fragment labels in the same manner.

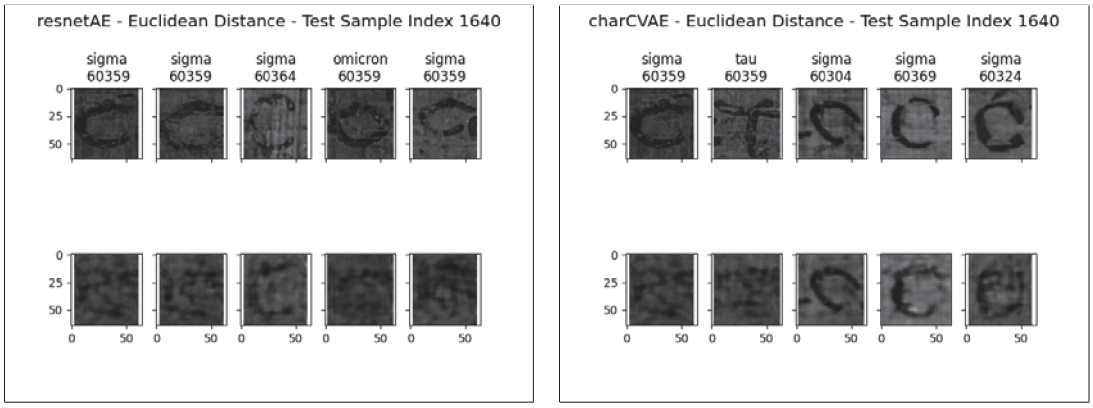
Euclidean Distance Measurements for the charCVAE

Starting with sample 1380 and sample 1640. In the resnetAE we have seen that in these two examples the character and the fragment labels are learned very well and cluster around each other in the latent space. For sample 1380 we have a phi that could be eliminated and for sample 1640 there is an omicron, that should be further away in the latent space. The hypothesis we claimed was, that with the conditioning on the labels, the unsupervised learning will transfer to a supervised learning and the model has more information and should therefore also learn the labels instead of only relying on the reconstruction. When looking at Figure 5.9 we can see that in comparison to the resnetAE we now have 4 sigmas closest to each other, no more omicron but an omega. In contrast to this rather good conditioning we can see in Figure 5.10 that the conditioning completely failed there. Five different character labels are closest to each other in the latent space conditioned on character labels.



(a) resnetAE (b) charCVAE

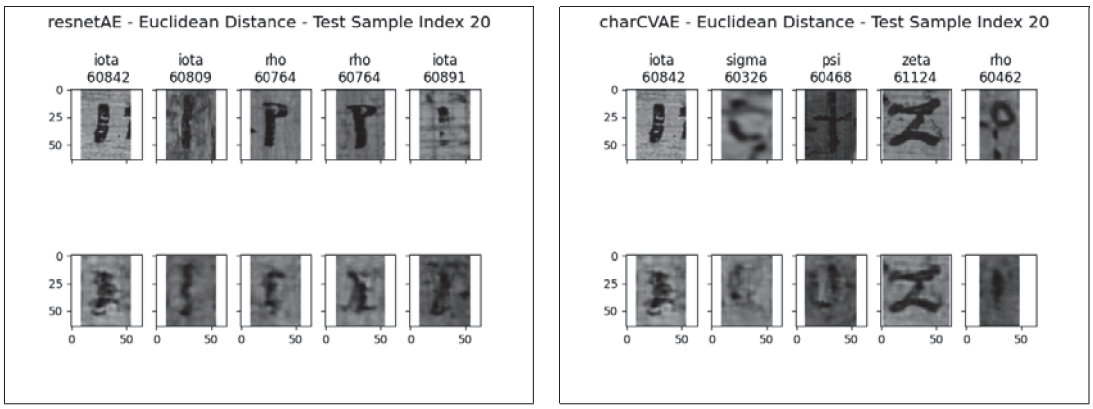
Figure 5.8: Sample 1380 - resnetAE and charCVAE



(a) resnetAE (b) charCVAE

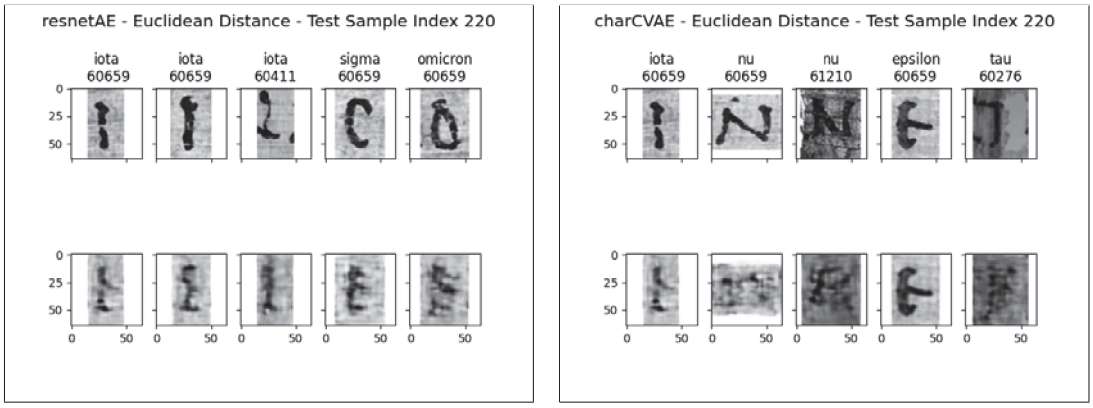
Figure 5.9: Sample 1640 - resnetAE and charCVAE

Further, we have seen previously that the letter iota is learned very well without any knowledge about the label. Some samples e.g. sample 20 and sample 220 still have had neighbors of a different class label but with similar features that should be corrected by conditioning. Once more, we have not the expected behaviour in the latent space. The neighboring seems poor and also introduces character labels that definitely should not be there, as we can see in Figure 5.10 and 5.11.



(a) resnetAE (b) charCVAE

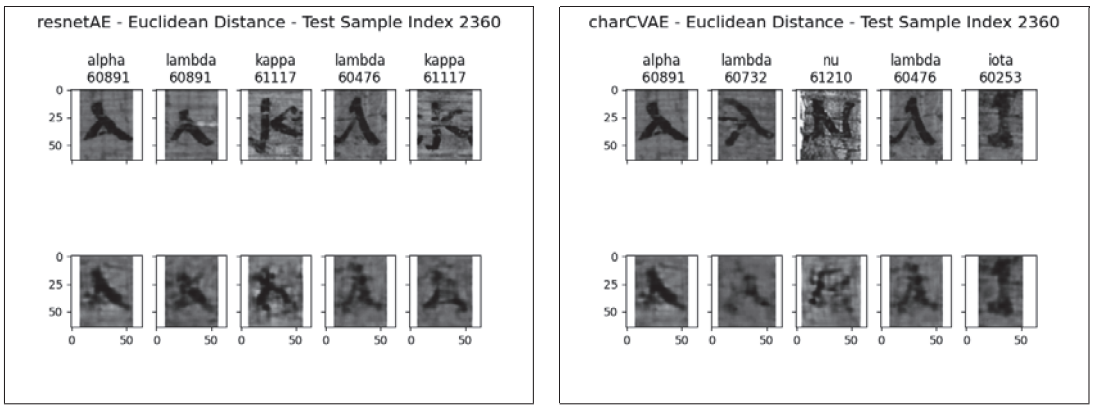
Figure 5.10: Sample 20 - resnetAE and charCVAE



(a) resnetAE (b) charCVAE

Figure 5.11: Sample 220 - resnetAE and charCVAE

Next, we analyze the characters alpha, delta and lambda. Previously, we stated that all three label classes share the triangular base feature when looking at them. Further, conditioning on the class label should eliminate the ambiguity between alpha, delta and lambda since the model knows the ground truth and is not only relying on the reconstruction to adapt its weights. Unfortunately, a glance at Figure 5.12 approves on the general behaviour too. Here again, a new class label that does not share any feature is introduced, namely the nu. Unfortunately, when looking at all other samples in the Appendix under A.3.3 this is just a lucky punch or outlier.



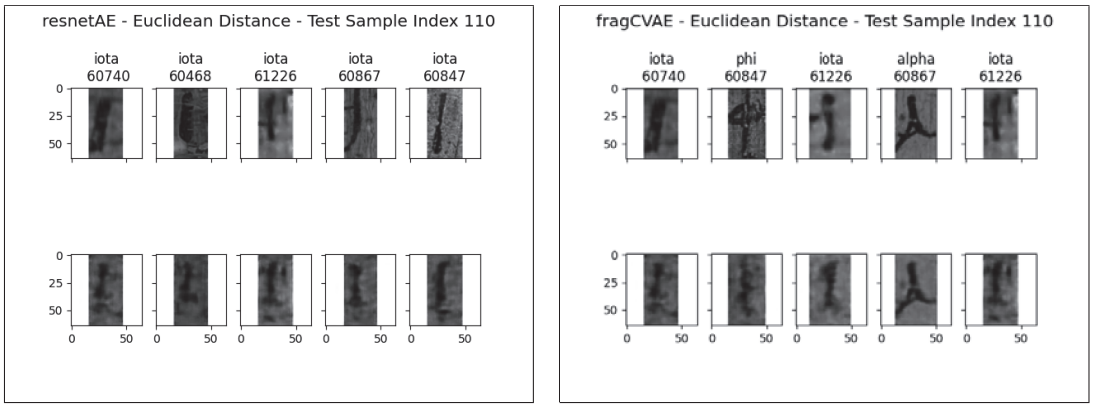
(a) resnetAE (b) charCVAE

Figure 5.12: Sample 2360 - resnetAE and charCVAE

Euclidean Distance Measurements for the fragCVAE

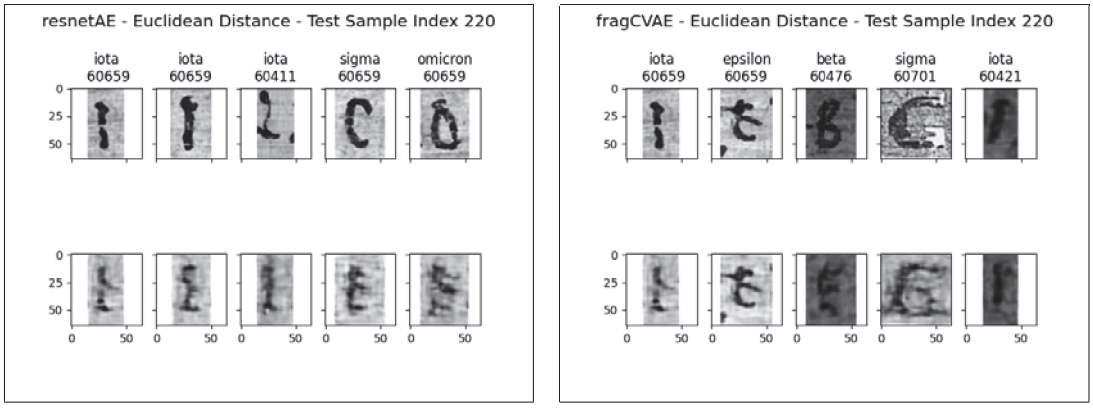
Since our data has two different labels, we chose to analyze the exact same approach again for the fragment labels. In Figure 5.13 we have seen that our resnet model is able to group iotas close to each other in the latent space. But, in this exact sample, we see nothing about the closeness

of the fragments. Again, our hypothesis is the same as for the charCVAE, just on the fragment labels. Meaning, if we condition the latent space on the fragment labels, we should achieve closeness of fragments in there euclidean distance plots. As we can see in Figure 5.13a this is not the case. There is not even one matching fragment label to the base sample. Further, looking at Figure 5.14a we already have a good distance to other iotas as well as different character class labels with the same fragment. Once more, the conditioning does not behave as expected. It introduces another time new fragment labels close to our base sample, therefore not shaping the latent space in the expected manner. The same behaviour can be examined in Figure 5.15 and 5.16 where we have already rather good closeness of fragment labels in the resnetAE, whereas the conditioning does not add any new information. In contrary, the learning and therefore closeness on fragment labels gets even worse.



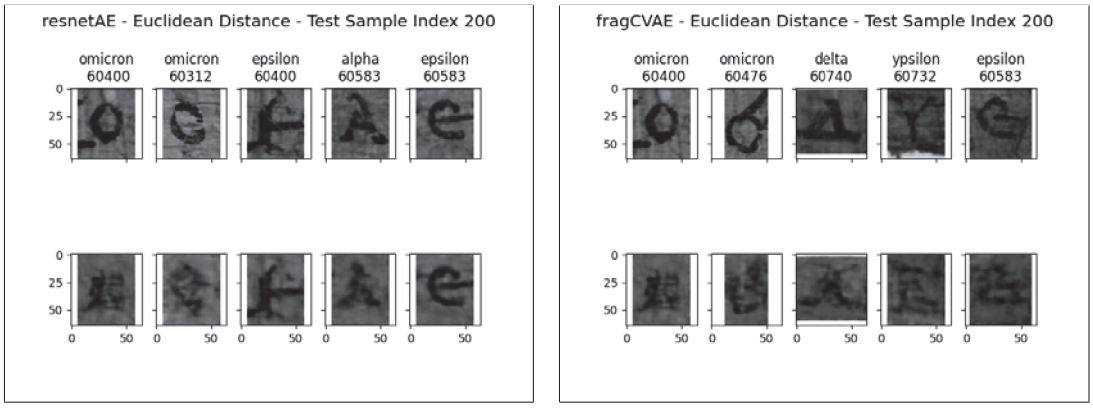
(a) resnetAE (b) fragCVAE

Figure 5.13: Sample 11 - resnetAE and fragCVAE



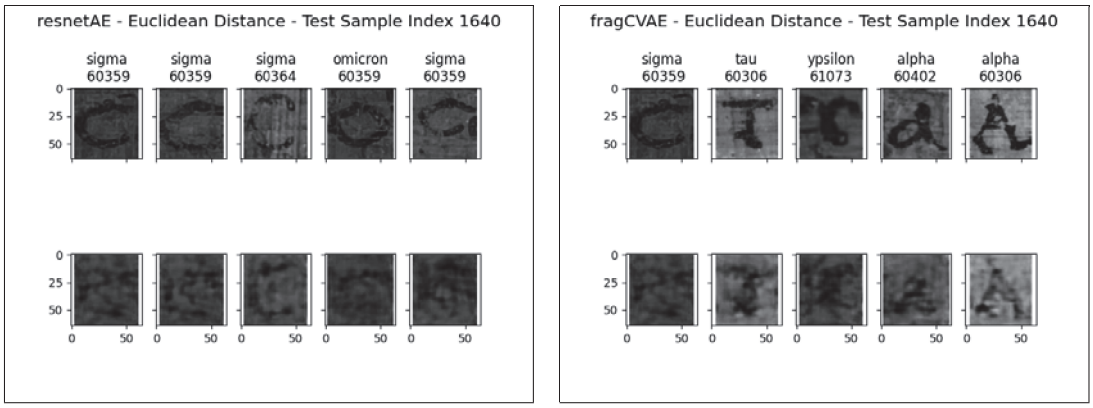
(a) resnetAE (b) fragCVAE

Figure 5.14: Sample 22 - resnetAE and fragCVAE



(a) resnetAE (b) fragCVAE

Figure 5.15: Sample 20 - resnetAE and fragCVAE



(a) resnetAE (b) fragCVAE

Figure 5.16: Sample 164 - resnetAE and fragCVAE

Clearly, the conditioning did not work as expected, whether for the character label conditioning nor the fragment label conditioning. This can have many possible causes but determining them needs further testing. This testing can be on a simple side by for example stacking more layers and adjusting the dimensionality of these layers as well as the resulting latent space. However, the cause may route even deeper. Perhaps the architectural choices of the resnetAE are already the limiting factor, for example the choice of the 48-dimensional latent space. At this point the number of variables that can be tweaked and fine-tuned is already immense. We will discuss further steps and ideas in the next Chapter.

6
Future Work

During the thesis we made many assumptions on the data as well as the possible improvements and adjustments of the already made approaches. This section is dedicated to take these assumption into further detail and analyze the possible changes that need to be made to improve the style interpretation. Also, we want to answer some questions that may have emerged during the thesis.

6.1 Dataset

Possibly the biggest variable of the whole equation for this analysis is the data and the dataset itself. Important factors to consider are the quality of the data available and the pre-processing that may be necessary. Throughout the thesis, we always looked at the whole dataset. We trained and searched for a model that can interpret everything at once, comparable to a top down approach. Effectively, we do not know if this is the right way to tackle the dataset. An idea omnipresent throughout the this was to approach the dataset with a bottom up approach instead of the top down approach made in all model approaches. What we are meaning by that is divide and conquer the dataset. It is much easier to only analyze one single label class for example all alphas or the most frequent fragment label. Then, in a next step join more data into the analysis dataset.

Having a smaller overall analysis dataset makes the evaluation easier and more comprehensive. As we have seen our testing set had a sample size of over 2000 observations. If we would split down the analysis dataset on only all alphas the testing set would have only 227 samples, which is a much more manageable size. Further, new evaluation possibilities will emerge. Having a model that encodes a latent space on only alphas opens the analysis for the fragment labels. In such a latent space we could calculate the similarity of the actual fragment clustering defined by the ground truth and the k-means clustering. This is much more congruent to our hypothesis stated.

6.2 Sophisticated Pre-Processing

Apart from how we dealt with the dataset as a whole, how we deal with what is inside the data needs to be improved. We tested pre-processing steps and introduced a simple pre-processing pipeline. As a matter of fact, this pre-processing is enough for a first touch with the data and get some models running. However, data loss is still present. Actually, as the dataset is at the moment, data loss is inevitable. The variety of resolutions throughout the dataset requires re-scaling, which always introduces dataloss. Actually the dataset is manually annotated by experts and therefore can be retaken or expanded. A future step to take for this sample generation is to have a fixed resolution size for the extraction of a cliplet out of a fragment. This improves overall quality of the dataset and makes working with the data much easier. Nevertheless, improvements can still be made to the current state, because a sophisticated background elimination is desirable. We explored binarization as one tool to eliminate the background but did not incorporate it into the pre-processing since it is too laborious and error prone. Further, the padding approach is a good base line to counter the variance in resolutions. However, the padding as it is right now introduces errors to the loss function and therefore the training, as can be seen in Figure 6.1.

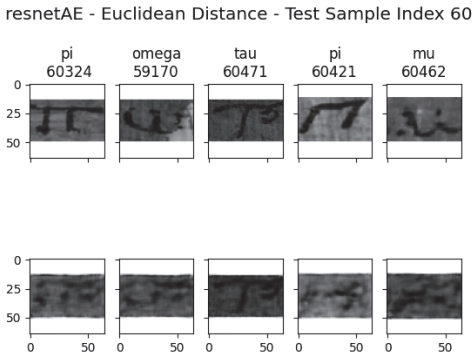


Figure 6.1: Euclidean Distances of Sample 60

Here, the padded white pixels take up to have of the whole image and are learned. Hence, the variety in labels and similarity in reconstruction. When the images have an extreme difference in height and width this is learned as well. This is also the cause that iotas get learned very well and can be mistaken with other "vertical" characters such as roh. Actually, masking the padded pixel or better said only calculate the loss of the identity of the cliplet will counter this behaviour. This is less of a pre-processing choice rather than a training optimization, which leads us to the next section.

6.3 Hyperparameter Tuning

Throughout the course of the thesis many design-related and architectural choices were made, some of which worked better than others. However, as we proceeded with each approach, the previous approach became clearer on the drawbacks, improvements and design choices that could have lead to different outcomes. The best example is the choice of the 48-dimensional latent space in the resnetAE. This was just arbitrarily chosen with no further thinking. As already stated in the two-step approach, this turned out to be a rather bad choice. For future steps in the style analysis it is desirable to go back one or two steps and rethink parameter settings. Anyhow, the optimization possibilities are immense. For the residual auto-encoder we can adjust the layer and also the latent space dimensionality in so many different ways. Further, fine-tuning is definitely needed for the two step CVAE approach. Perhaps more fully connected layers will fix the conditioning. Given that the exact origin of the problem is now known, we cannot bypass taking steps back.

6.4 Full Residual CVAE

As we have seen in the section 5.1.2, the two step approach on conditioning the latent space has not worked as expected. Regardless, we should not give up the idea on conditioning the model on the labels. Taking a step back is required to analyze and fine tune the residual auto-encoder. Additionally, instead of only fine tune the residual auto-encoder and then expand the architecture of the two step approach, we could directly incorporate the conditioning into our resnetAE. Therefore, we would form a full residual CVAE architecture that combines all advantages of the different approaches explored. In theory this is a very powerful model due to the robustness of the residual blocks and the feature learning over conditioning and reconstruction. This model can be placed as a shadow model when adjusting the original resnetAE to reassure the architectural choices made.

7
Conclusion

As we have seen in the course of the thesis, we had to deal with a challenging dataset. With pre-processing noise in the dataset needs to be eliminated. Further, it contains a variety of different resolutions that need to be brought into a uniform format. However, as we have stated a minimal pre-processing pipeline, this needs to be improved for further analysis. In addition, a hypothesis has been formed about style and style analysis. To answer this hypothesis we have explored three approaches with auto-encoder architecture. We tried to augment a basic convolutional auto-encoder model with residual learning that gave a broader understanding about the dataset and the data within. This understanding was created by analyzing the latent space of the auto-encoder with commonly known techniques. However, not all techniques yielded the expected results or could completely explain the models choices on feature learning. Also the hypothesis was not answered or confirmed in a respectable manner. In fact, it turned out that the hypothesis about style is much more complicated and complex to answer as a whole. However, the analysis showed that taking the whole dataset for training and analysis is overwhelming. Isolating manageable portions of the dataset is a better and more understandable approach for the style analysis.

Hence, in a next step we tried further augmentations with a two step approach. By conditioning the latent space of our residual learning on the labels given in the dataset, ambiguity in the style analysis should have been eliminated. However, the opposite was the case. The conditioning introduced even more ambiguity and unexplainable behaviour. We have not yet found a reasonable explanation for this failed conditioning. Therefore, for further inspection and improvements to the overall style analysis we have to take a step back. We propose that either the conventional residual auto-encoder undergoes some additional inspection by fine-tuning its layers and parameter alone. As an alternative, one could merge the two step approach into a single model thus combining all advantages of the made approaches and therefore resulting in conditioning the whole training rather than only the already formed latent space.

Concluding, the hypothesis for style has not been confirmed and further work needs to be done to

profoundly make assumptions on style and writer identification. However, the thesis has shown and confirmed the advantages of using auto-encoders as the network architecture of choice. In addition, we now know how to deal with the data and are therefore ready for these further steps. By taking some steps back and rethink the approaches and change the top down view on the data to a bottom up approach, we can divide and conquer the style analysis.

Bibliography

[1] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

[2] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.

[3] Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Larry D Jackel, Yann LeCun, Urs A Muller, Edward Sackinger, Patrice Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5)*, volume 2, pages 77–82. IEEE, 1994.

[4] Jorge Calvo-Zaragoza and Antonio-Javier Gallego. A selectional auto-encoder approach for document image binarization. *Pattern Recognition*, 86:37–47, 2019.

[5] Vincent Christlein, Martin Gropp, Stefan Fiel, and Andreas Maier. Unsupervised feature learning for writer identification and writer retrieval. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 991–997. IEEE, 2017.

[6] Joseph T Colonel and Sam Keene. Conditioning autoencoder latent spaces for real-time timbre interpolation and synthesis. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2020. doi: 10.1109/IJCNN48605.2020.9207666.

[7] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[8] Ricard Durall, Kalun Ho, Franz-Josef Pfreundt, and Janis Keuper. Latent space conditioning on generative adversarial networks, 2020. URL https://arxiv.org/abs/2012.08803.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

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

[12] Dmytro Kotovenko, Artsiom Sanakoyeu, Sabine Lang, and Bjorn Ommer. Content and style disentanglement for artistic style transfer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4422–4431, 2019.

[13] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[14] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018. URL https://arxiv.org/abs/1802.03426.

[15] Sidra Nasir, Imran Siddiqi, and Momina Moetesum. Writer characterization from handwriting on papyri using multi-step feature learning. In *International Conference on Document Analysis and Recognition*, pages 451–465. Springer, 2021.

[16] Erik Norlander and Alexandros Sopasakis. Latent space conditioning for improved classification and anomaly detection, 2019. URL https://arxiv.org/abs/1911.10599.

[17] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/eb86d510361fc23b59f18c1bc9802cc6-Paper.pdf.

[18] Arshia Rehman, Saeeda Naz, Muhammad Imran Razzak, and Ibrahim A Hameed. Automatic visual features for writer identification: a deep learning approach. *IEEE access*, 7: 17149–17157, 2019.

[19] DE Rumelhart, GE Hinton, and RJ Williams. Miller g., & fellbaum c.(1991) semantic networks of english. cognition, 41, 197-229. millis, ml, & button, sb (1989) the effect of polysemy on lexical decision time: Now you see it, now you don't. memory and cognition, 17, 141-147. pinker, s.(1989) learnability and cognition. cambridge, ma: Mit press. pustejovsky, j.(1992) in b. levin & s. pinker (eds.) lexical and conceptual semantics. cam. *Science*, 12: 423–466.

[20] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11, 2014.

[21] Erich Schubert and Michael Gertz. Intrinsic t-stochastic neighbor embedding for visualization and outlier detection. In *International Conference on Similarity Search and Applications*, pages 188–203. Springer, 2017.

[22] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[23] Linjie Xing and Yu Qiao. Deepwriter: A multi-stream deep cnn for text-independent writer identification. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 584–589. IEEE, 2016.

A
Appendix

A.1 Additional Information on the Dataset

A.1.1 Exact sample sizes for the character labels

alpha $\alpha = 1137$, beta $\beta = 69$, chi $\chi = 162$, delta $\delta = 272$, epsilon $\epsilon = 1223$, eta $\eta = 440$, gamma $\gamma = 176$, iota $\iota = 1247$, kappa $\kappa = 363$, lambda $\lambda = 456$, mu $\mu = 391$, nu $\nu = 937$, omega $\omega = 320$, omicron $o = 1203$, phi $\phi = 143$, pi $\pi = 415$, psi $\psi = 15$, rho $\rho = 543$, sigma $\sigma = 867$, tau $\tau = 725$, theta $\theta = 218$, xi $\xi = 28$, ypsilon $\upsilon = 461$, zeta $\zeta = 34$

A.1.2 Exact sample count for the fragment labels

('60306', 774), ('60891', 753), ('60583', 740), ('61073', 540), ('61226', 402), ('60326', 375), ('60359', 358), ('61165', 320), ('60732', 310), ('60411', 296), ('60764', 278), ('60476', 269), ('61210', 266), ('60462', 265), ('60215', 230), ('60847', 227), ('60633', 218), ('60740', 209), ('60421', 207), ('60867', 182), ('60398', 178), ('60701', 174), ('61122', 162), ('60333', 150), ('66764', 148), ('61117', 146), ('60324', 139), ('60941', 135), ('60471', 128), ('60220', 127), ('61140', 127), ('60304', 123), ('60402', 110), ('61239', 110), ('60343', 101), ('61236', 96), ('61124', 96), ('60468', 91), ('60670', 87), ('60940', 86), ('60659', 86), ('60267', 84), ('60221', 82), ('59170', 79), ('61240', 74), ('60479', 66), ('61212', 61), ('60242', 59), ('60258', 51), ('60812', 51), ('61245', 49), ('60965', 49), ('60663', 49), ('60771', 48), ('60253', 47), ('61106', 46), ('60246', 42), ('60901', 41), ('61228', 40), ('60400', 39), ('60589', 39), ('60369', 37), ('60475', 37), ('61246', 36), ('60214', 36), ('60291', 36), ('61244', 36), ('61138', 36), ('60809', 35), ('61112', 34), ('60998', 33), ('60808', 32), ('60276', 31), ('60337', 31), ('60367', 31), ('60842', 29), ('60364', 29), ('61026', 28), ('60910', 27), ('60481', 27), ('60492', 26), ('60934', 26), ('60283', 25), ('60290', 25), ('60219', 24), ('61213', 22), ('60238', 21), ('60216', 21), ('60255', 19), ('60248', 18), ('60312', 18), ('60810', 17), ('60217', 16), ('60251', 15), ('65858', 14), ('61141', 12)

A.2 Additional Material on resnetAE Evaluation

A.2.1 resnetAE - Dimensionality Reduction

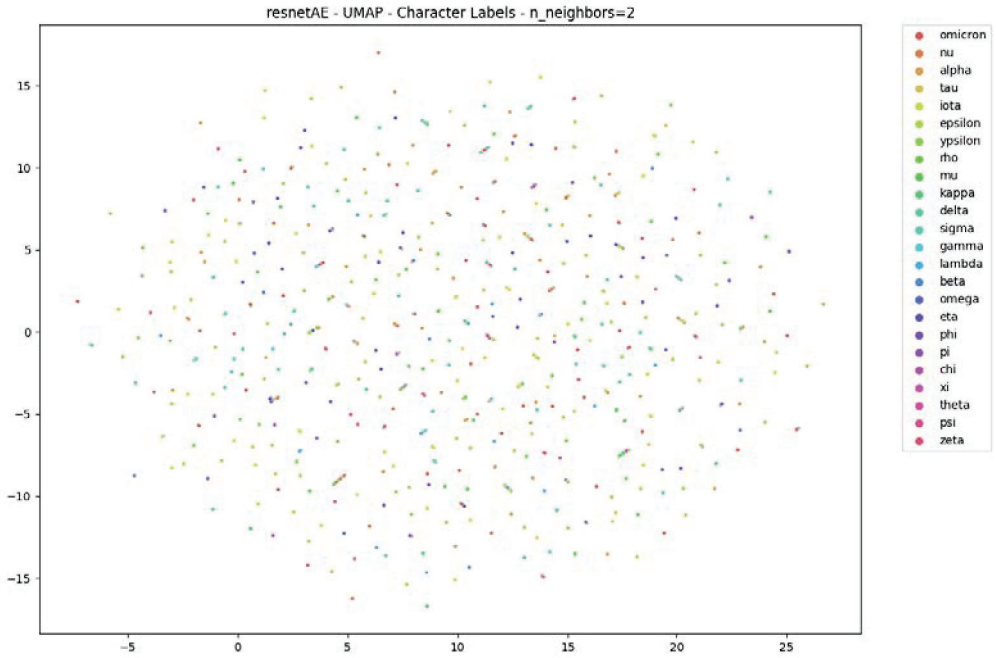


Figure A.1: UMAP Plot of the full 48-dimensional Space with 2 Neighbors

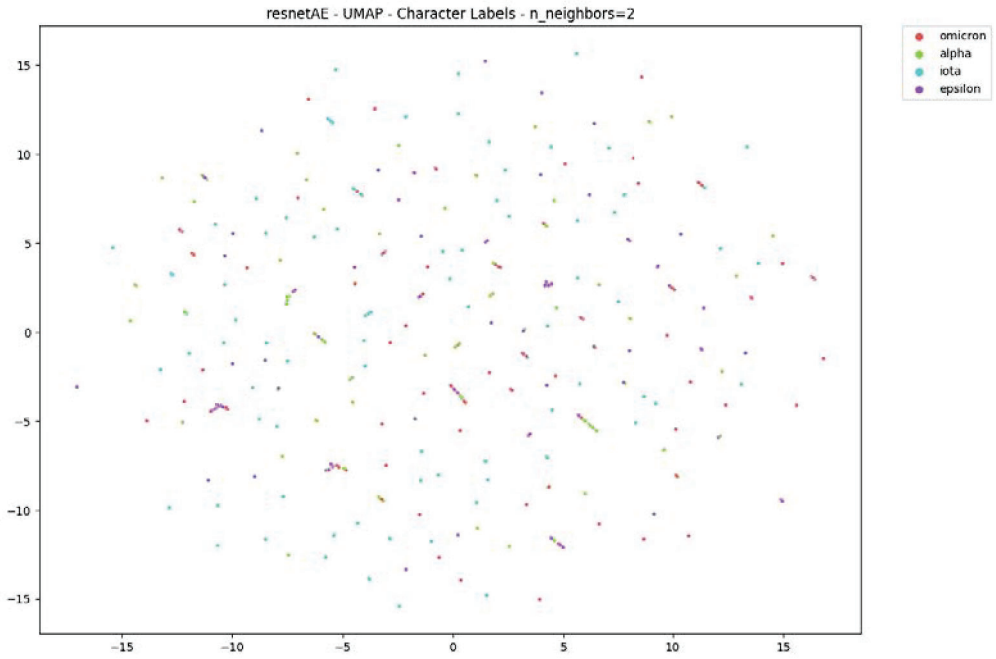


Figure A.2: UMAP Plot of the 48-dimensional Space of Characters alpha, epsilon, iota and omicron with 2 Neighbors

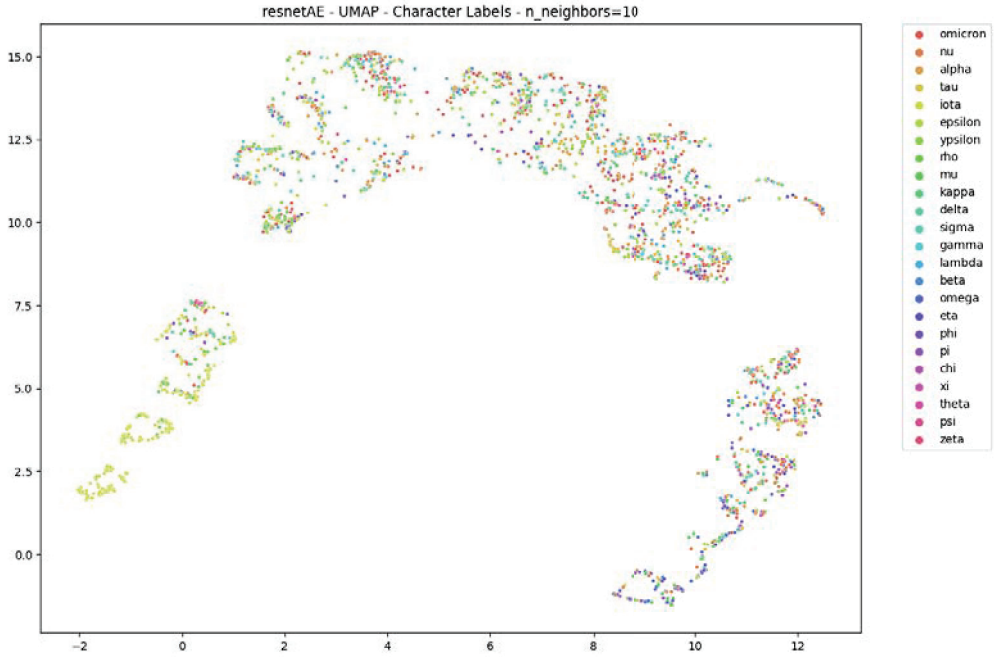


Figure A.3: UMAP Plot of the full 48-dimensional Space with 10 Neighbors

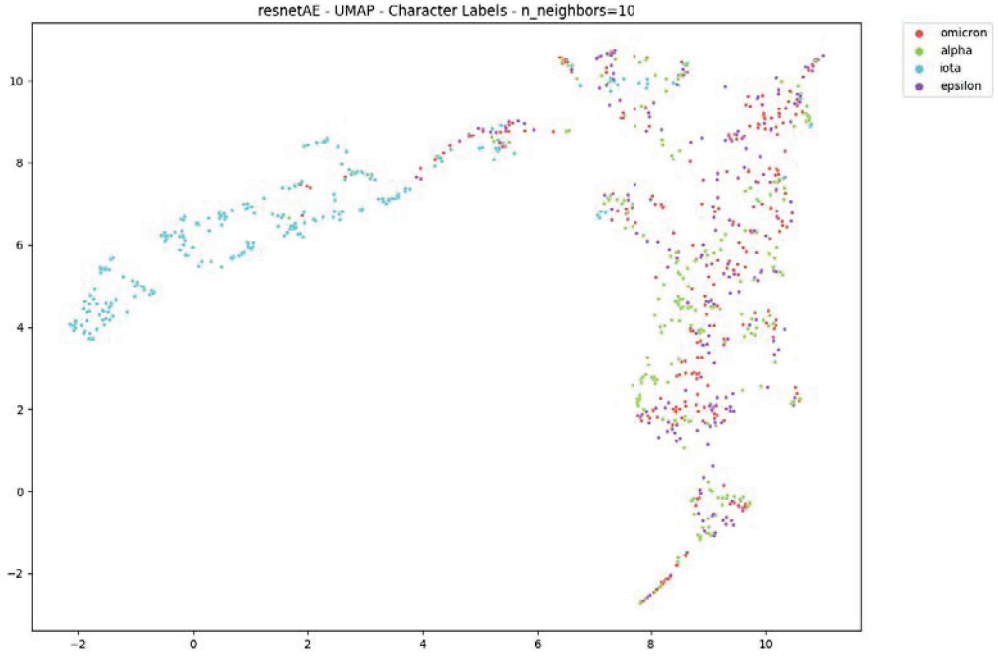


Figure A.4: UMAP Plot of the 48-dimensional Space of Characters alpha, epsilon, iota and omicron with 10 Neighbors

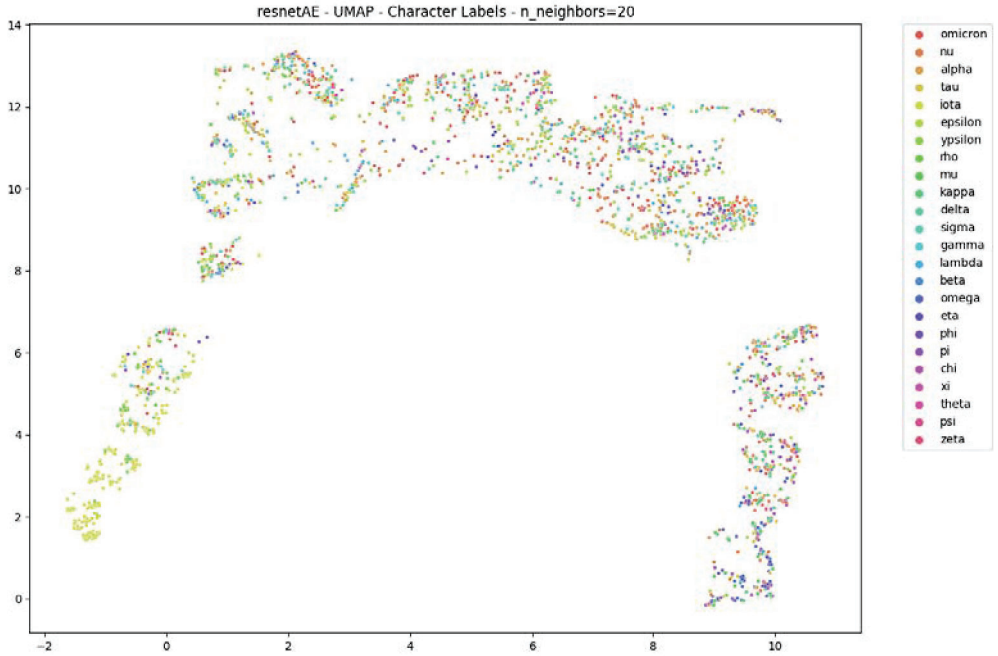


Figure A.5: UMAP Plot of the full 48-dimensional Space with 20 Neighbors

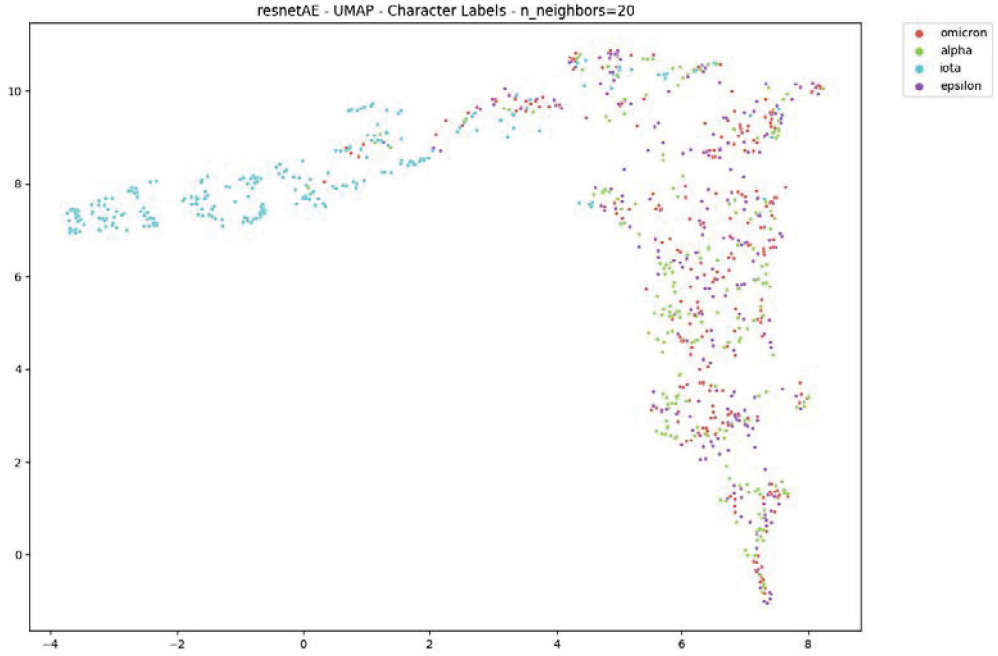


Figure A.6: UMAP Plot of the 48-dimensional Space of Characters alpha, epsilon, iota and omicron with 20 Neighbors

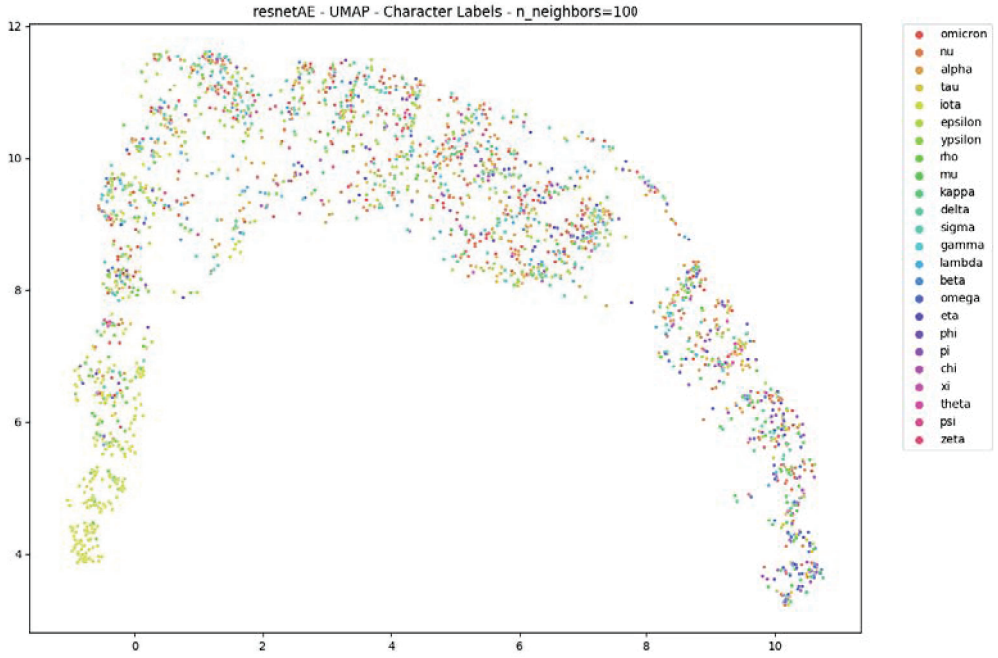


Figure A.7: UMAP Plot of the full 48-dimensional Space with 100 Neighbors

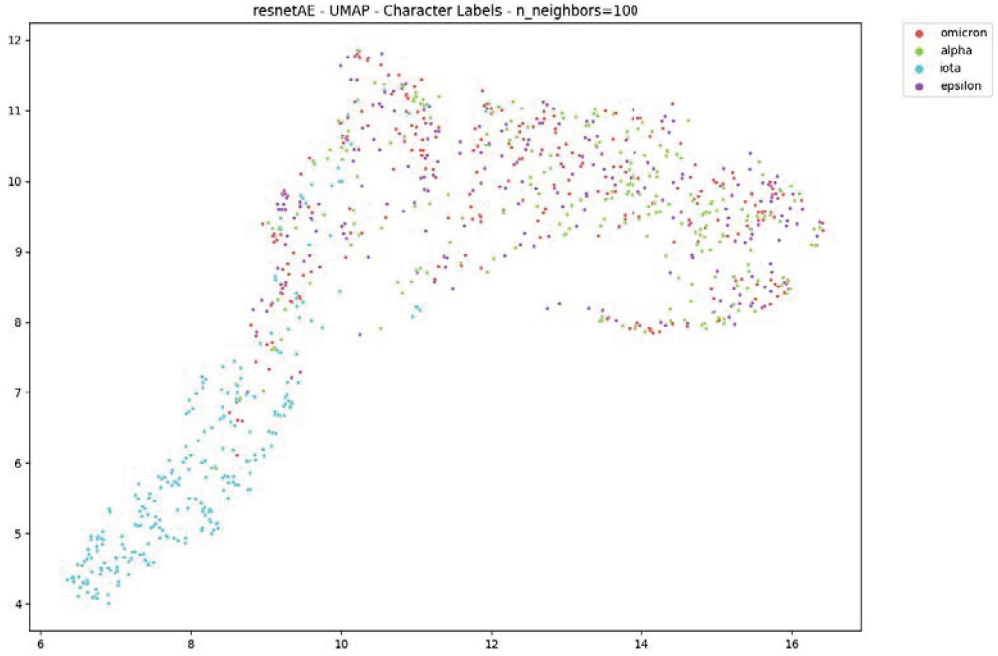


Figure A.8: UMAP Plot of the 48-dimensional Space of Characters alpha, epsilon, iota and omicron with 100 Neighbors

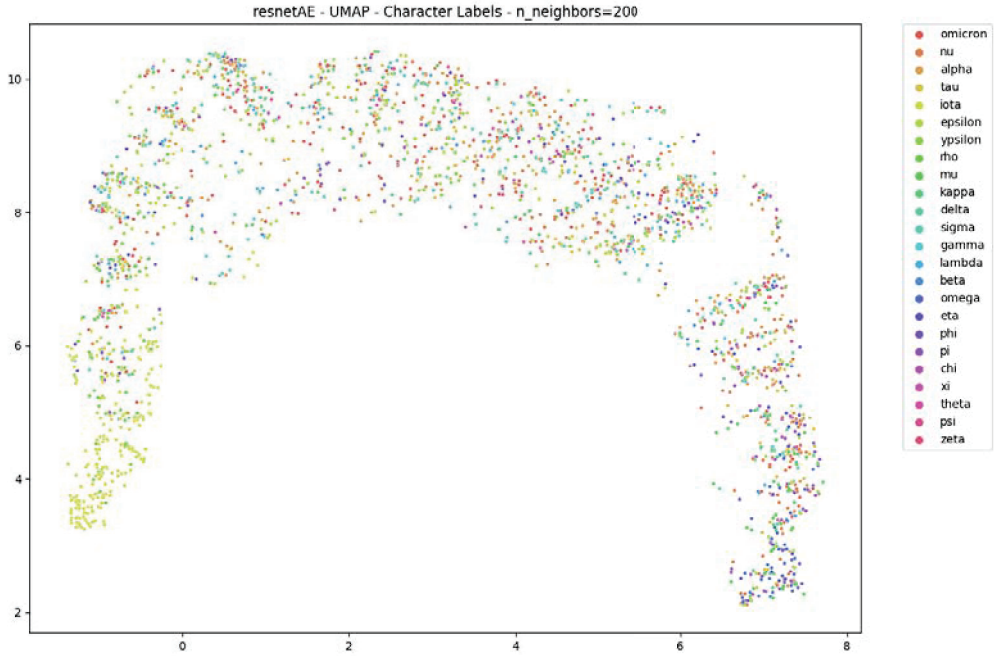


Figure A.9: UMAP Plot of the full 48-dimensional Space with 200 Neighbors

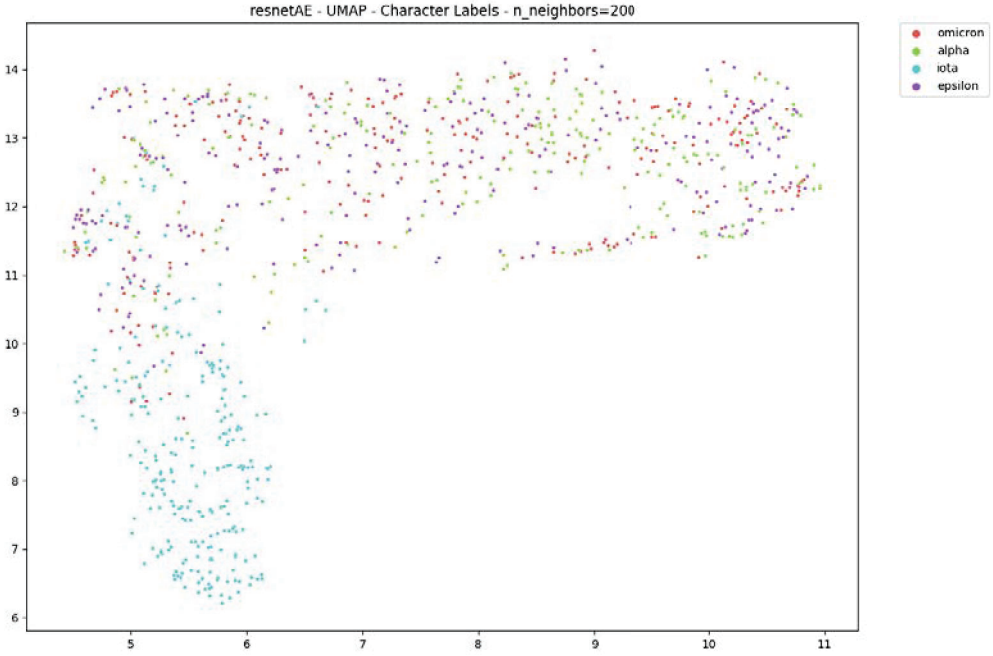


Figure A.10: UMAP Plot of the 48-dimensional Space of Characters alpha, epsilon, iota and omicron with 200 Neighbors

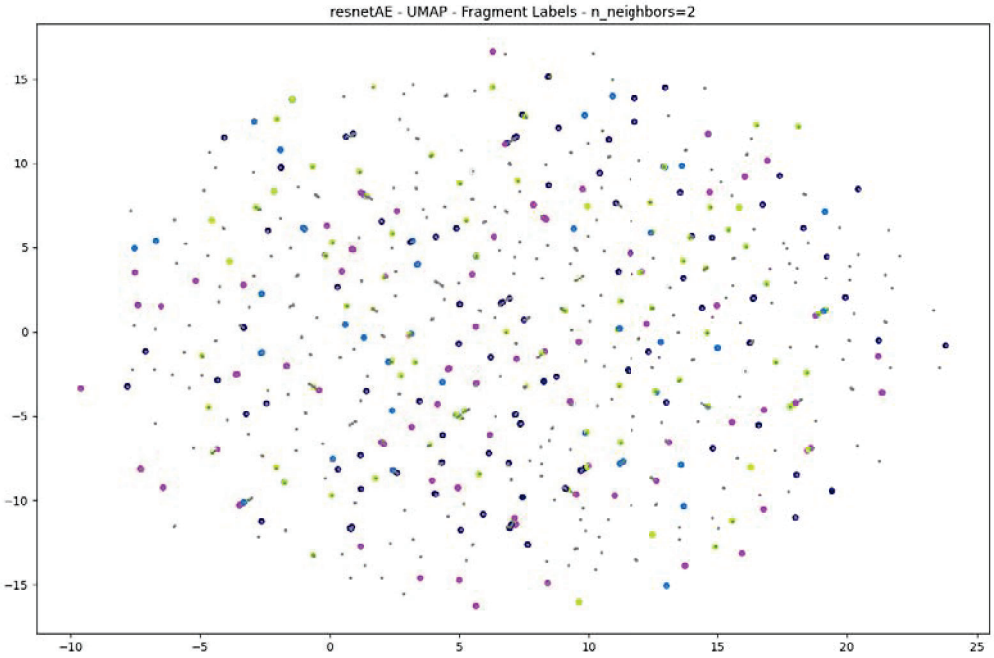


Figure A.11: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 2 Neighbors

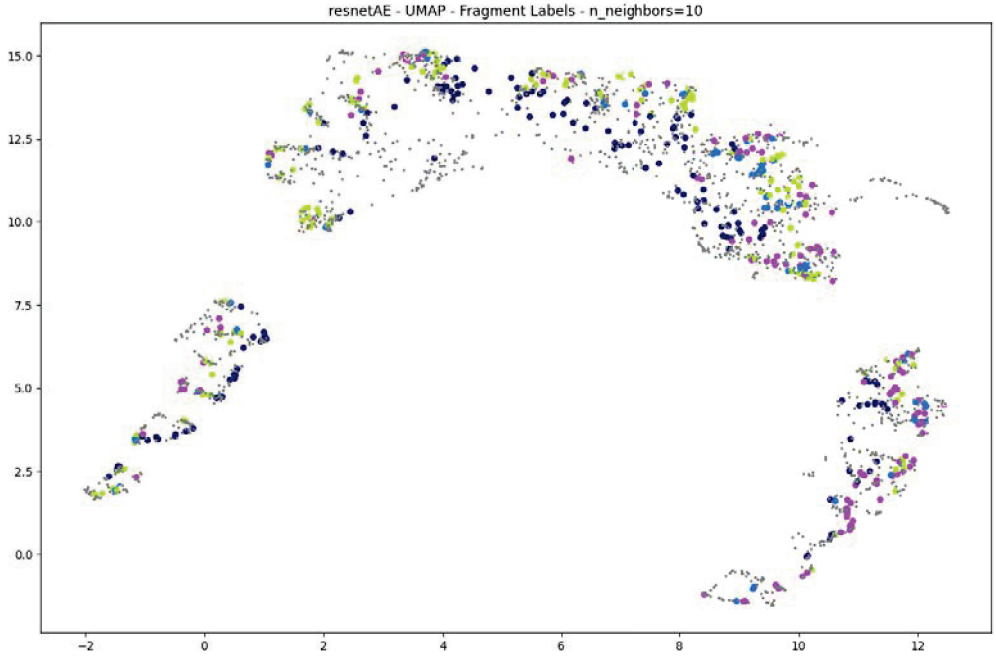


Figure A.12: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 10 Neighbors

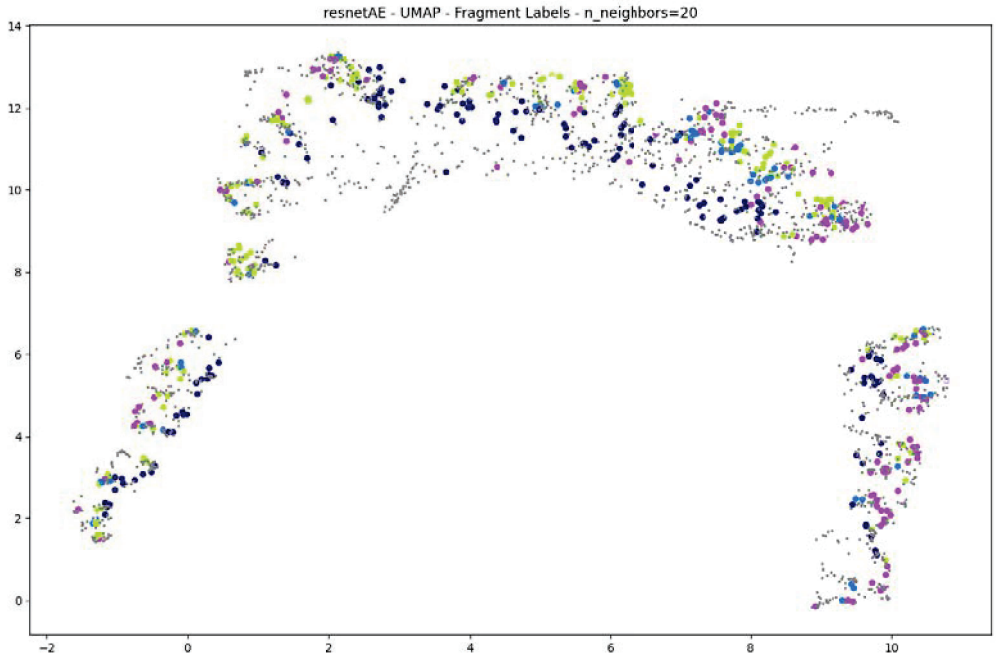


Figure A.13: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 20 Neighbors

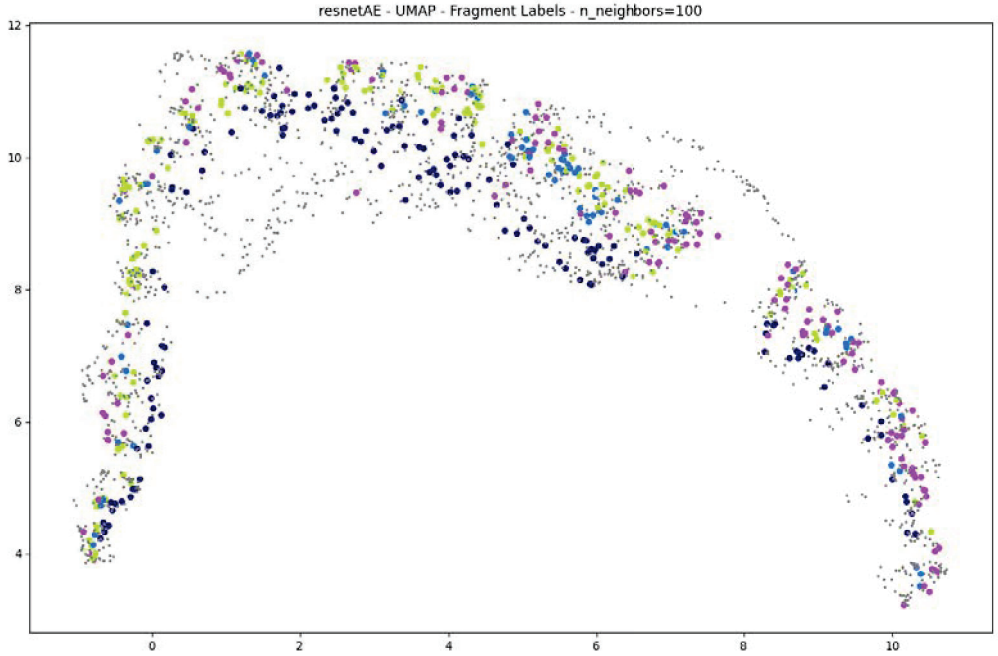


Figure A.14: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 100 Neighbors

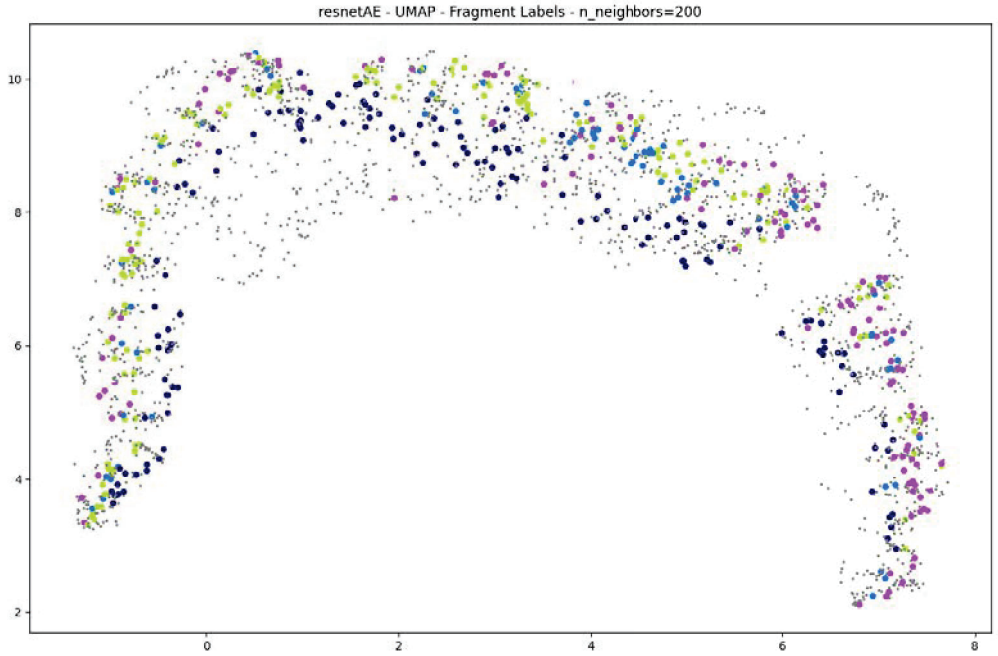


Figure A.15: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 200 Neighbors

A.2.2 resnetAE - Clustering and Rand Index

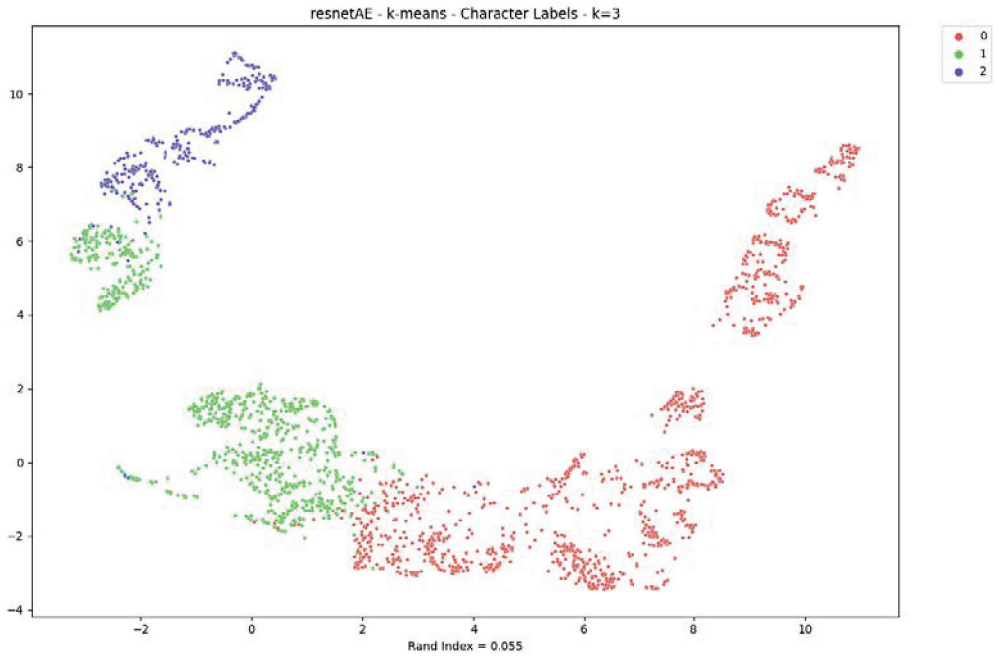


Figure A.16: K-Means with 3 Clusters on Full Latent Space

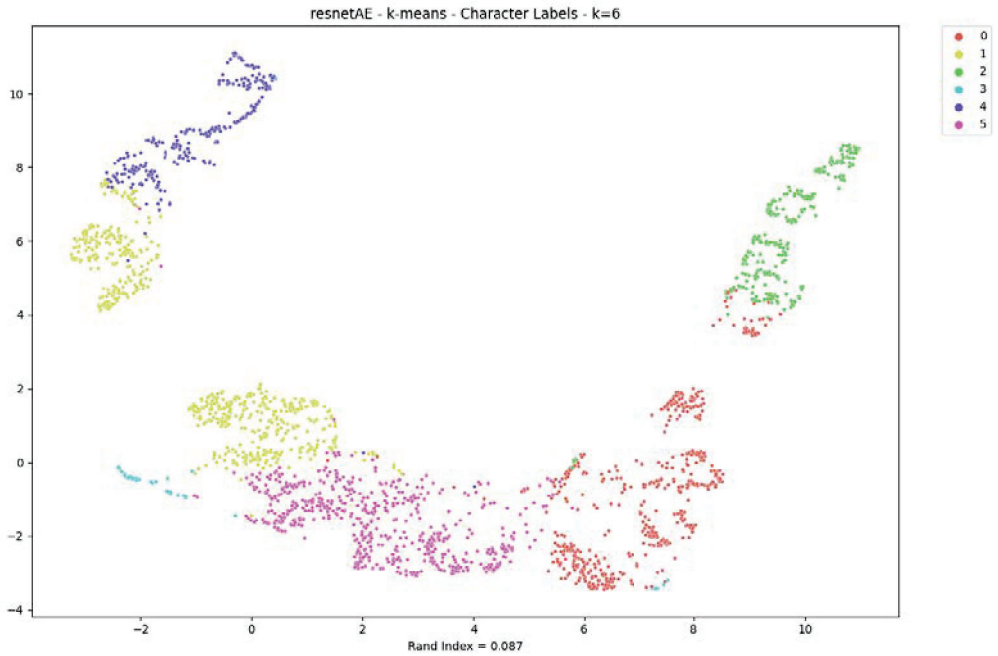


Figure A.17: K-Means with 6 Clusters on Full Latent Space

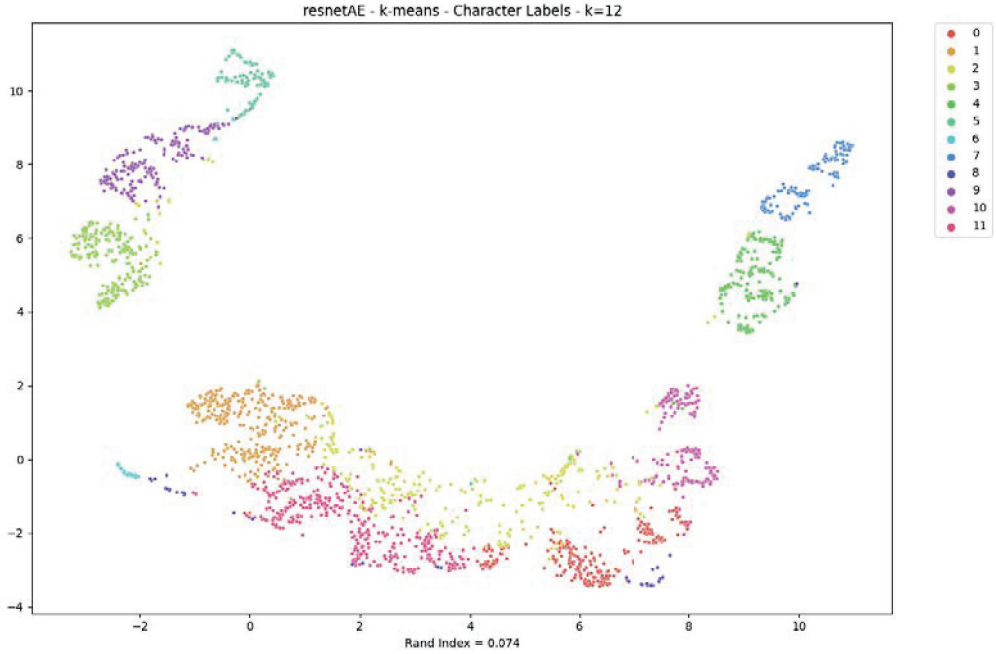


Figure A.18: K-Means with 12 Clusters on Full Latent Space

A.2.3 resnetAE - Euclidean Distance

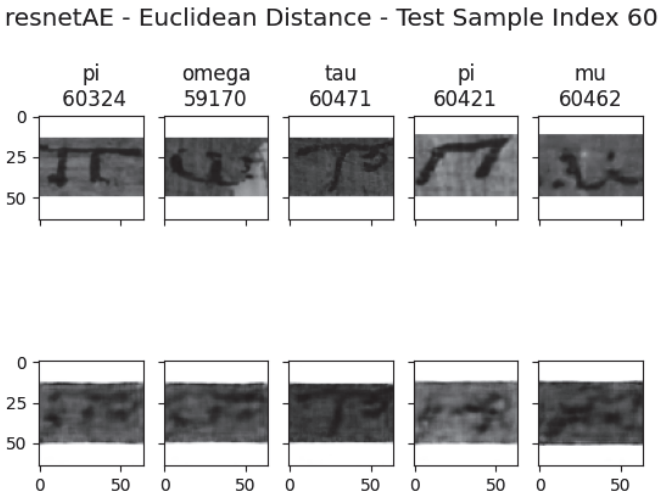


Figure A.19: Euclidean Distances of Sample 60

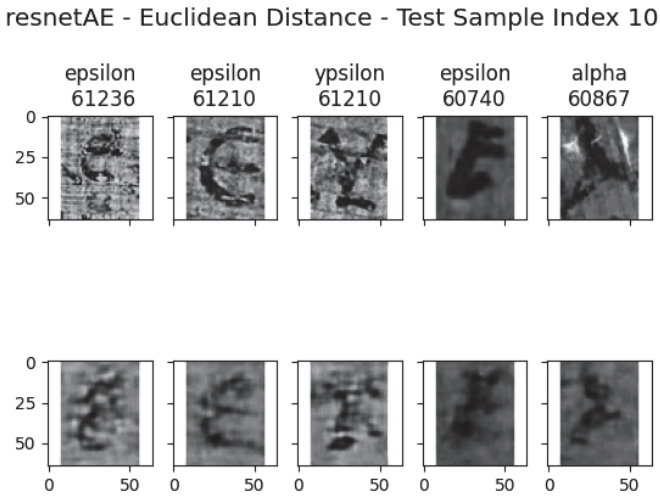


Figure A.20: Euclidean Distances of Sample 10

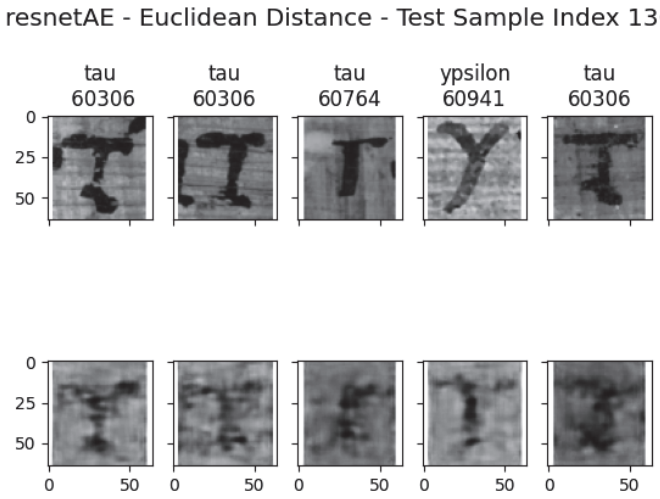


Figure A.21: Euclidean Distances of Sample 13

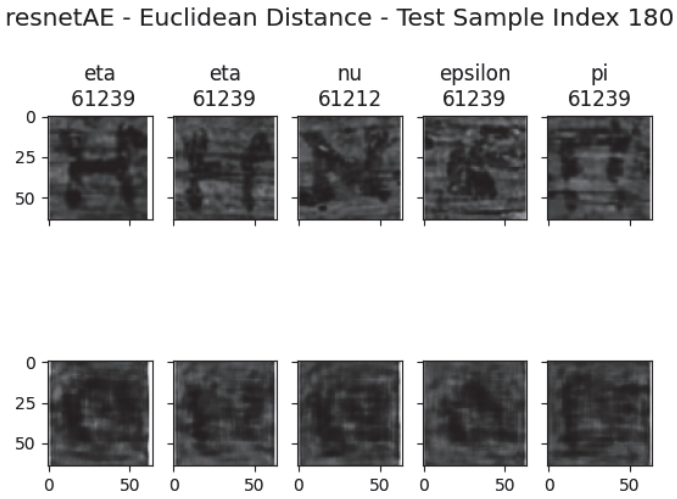


Figure A.22: Euclidean Distances of Sample 180

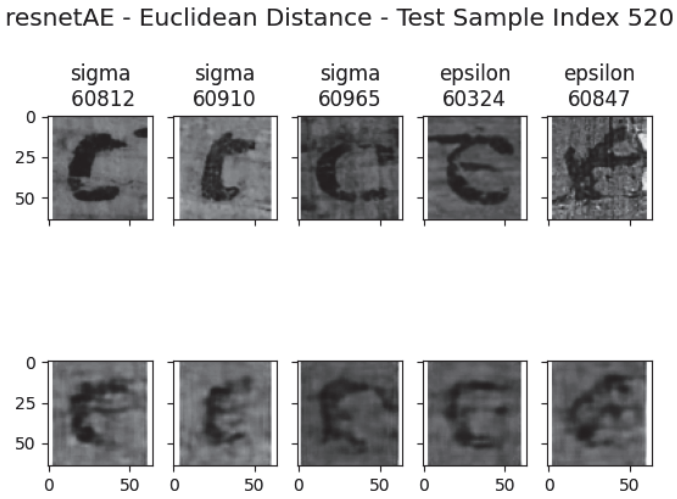


Figure A.23: Euclidean Distances of Sample 52

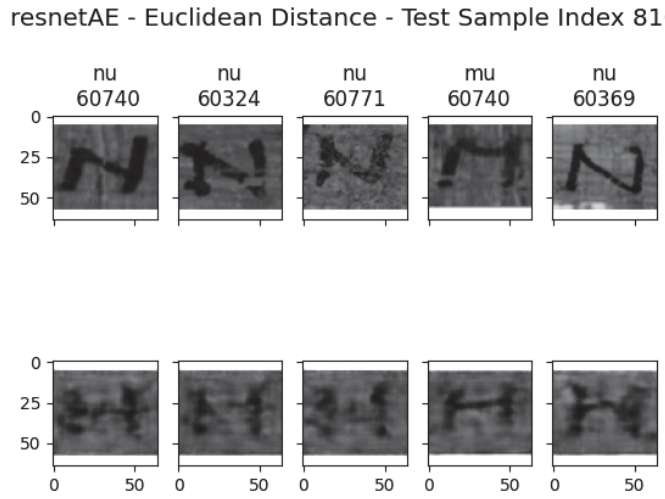


Figure A.24: Euclidean Distances of Sample 810

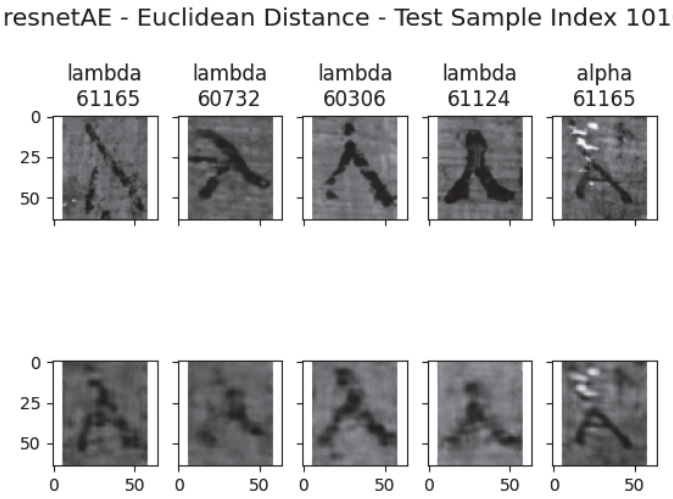


Figure A.25: Euclidean Distances of Sample 101

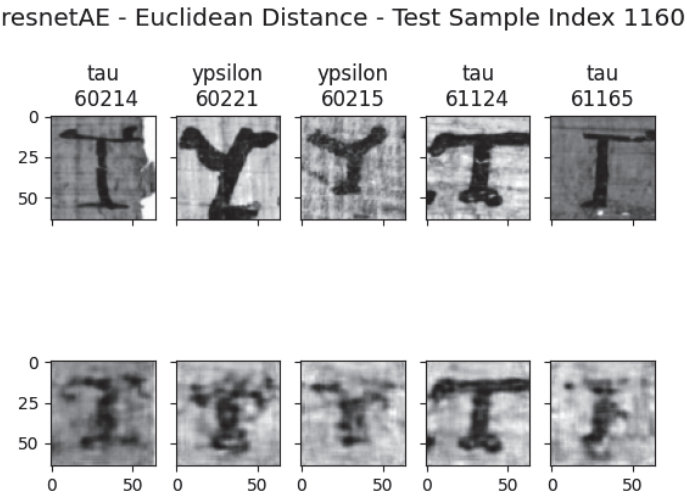


Figure A.26: Euclidean Distances of Sample 116

A.3 Additional Material on charCVAE Evaluation

A.3.1 charCVAE - Dimensionality Reduction

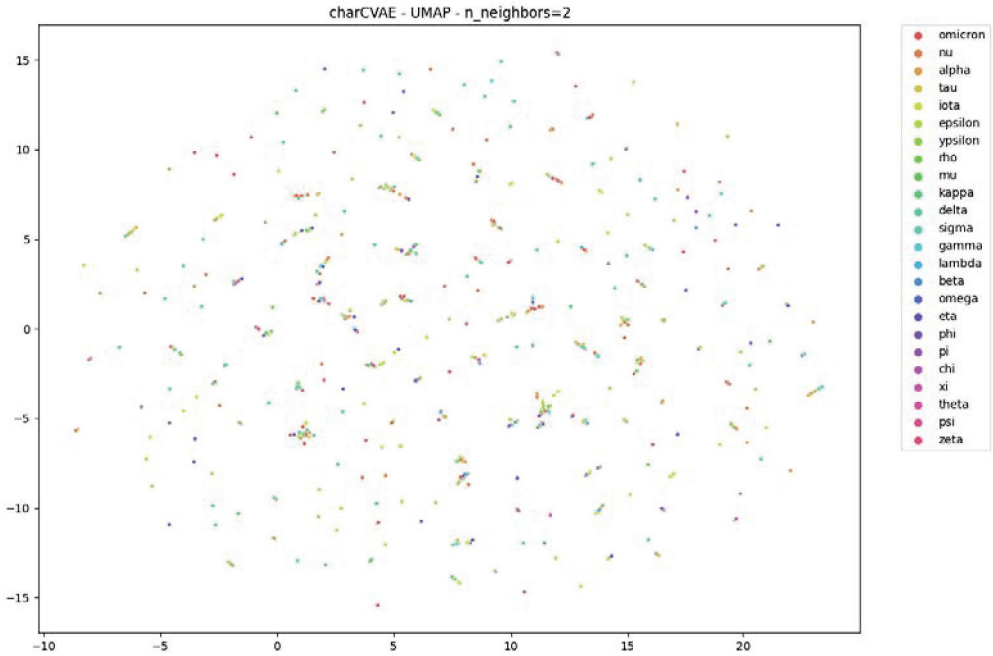


Figure A.27: UMAP Plot of the full 48-dimensional Space with 2 Neighbors

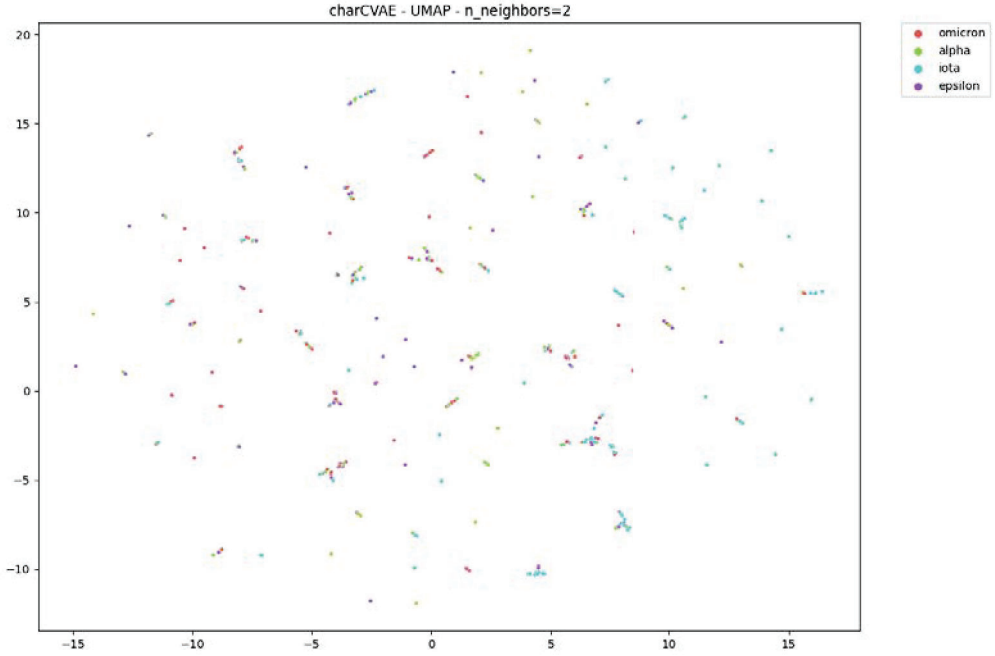


Figure A.28: UMAP Plot of the Latent Space of the most frequent Character Labels with 2 Neighbors

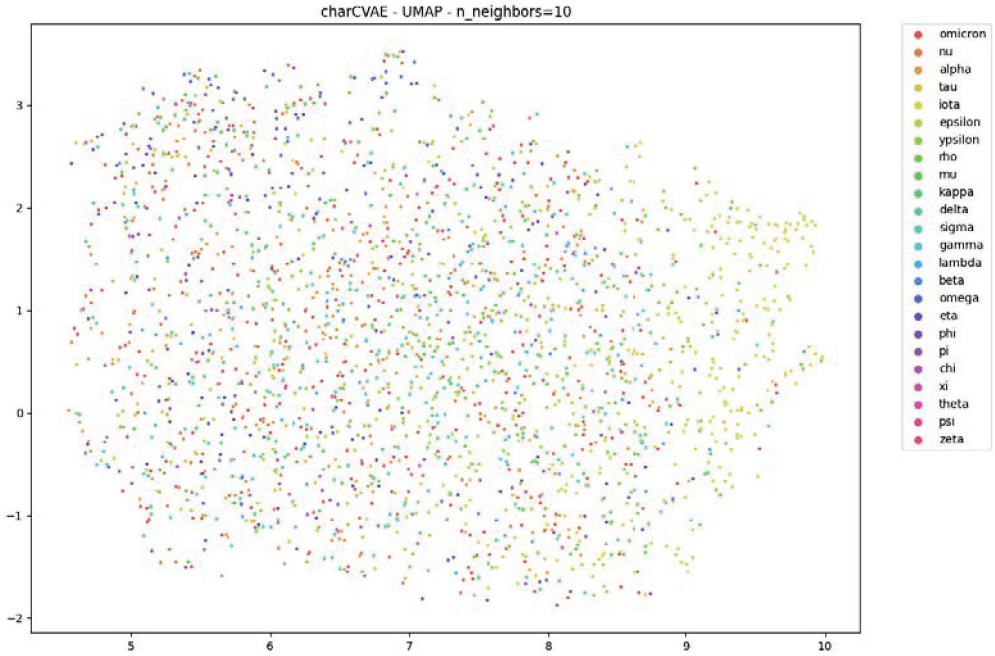


Figure A.29: UMAP Plot of the full 48-dimensional Space with 10 Neighbors

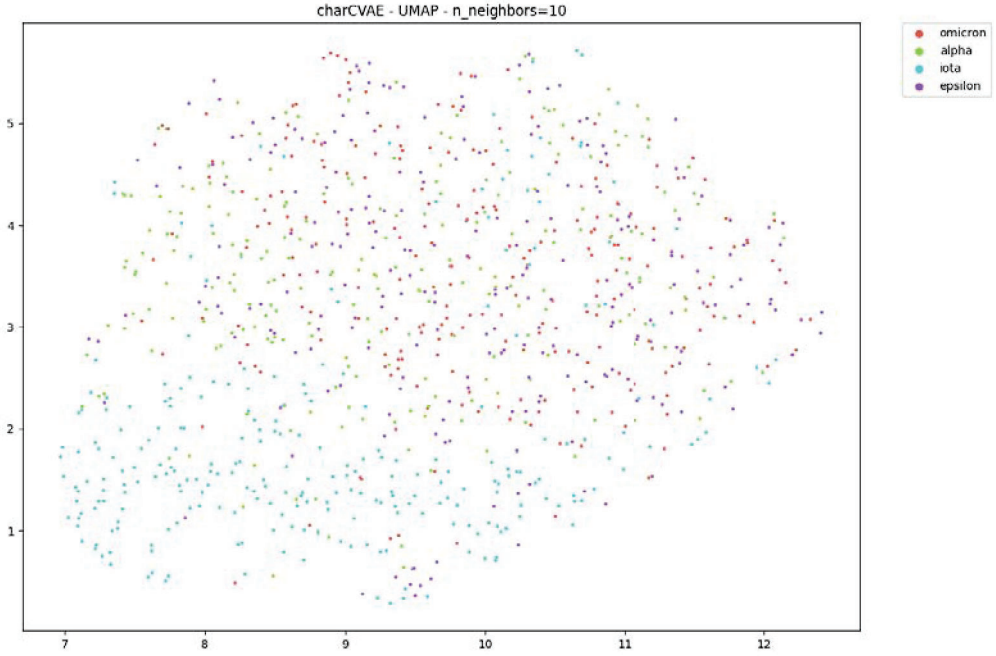


Figure A.30: UMAP Plot of the Latent Space of the most frequent Character Labels with 10 Neighbors

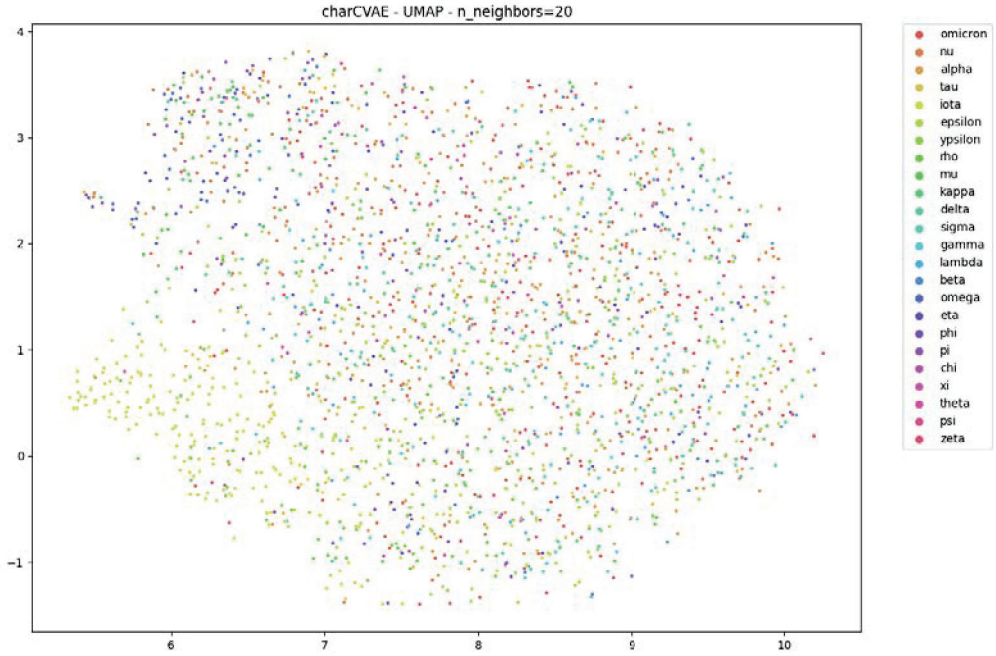


Figure A.31: UMAP Plot of the full 48-dimensional Space with 20 Neighbors

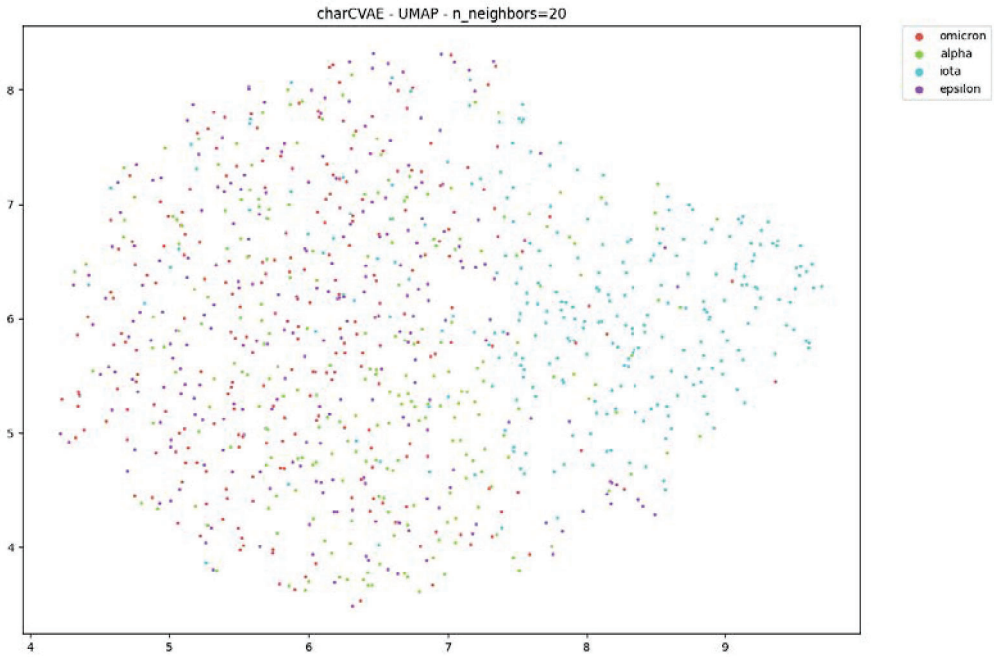


Figure A.32: UMAP Plot of the Latent Space of the most frequent Character Labels with 20 Neighbors

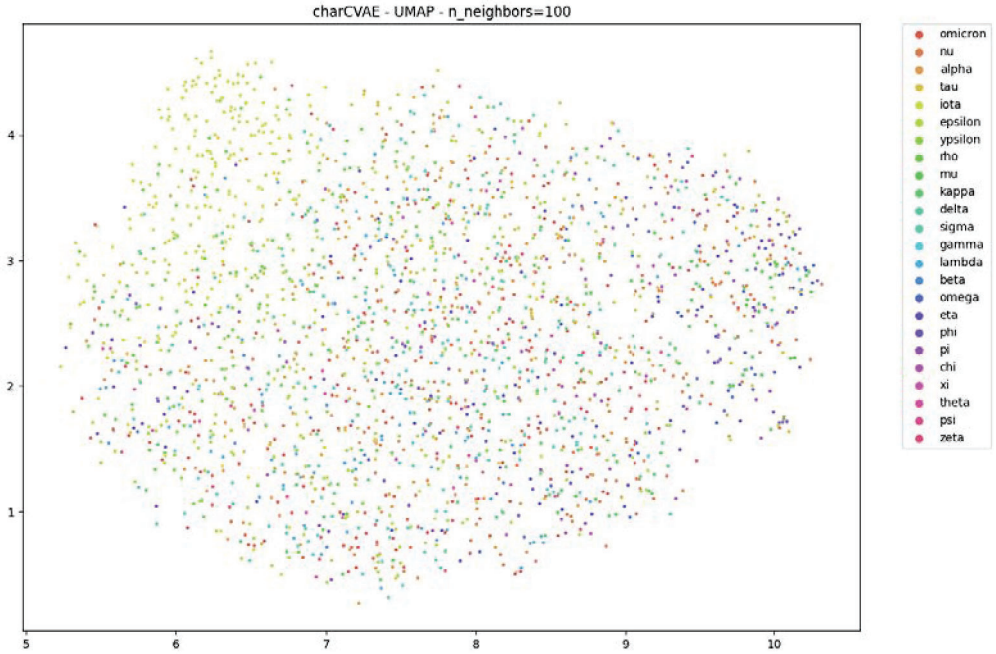


Figure A.33: UMAP Plot of the full 48-dimensional Space with 100 Neighbors

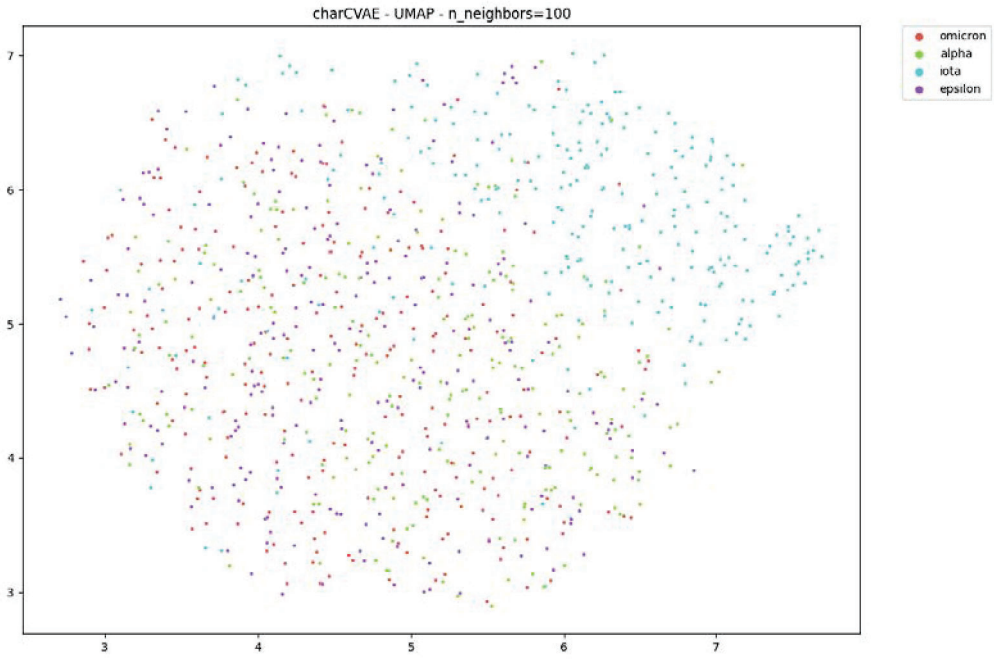


Figure A.34: UMAP Plot of the Latent Space of the most frequent Character Labels with 100 Neighbors

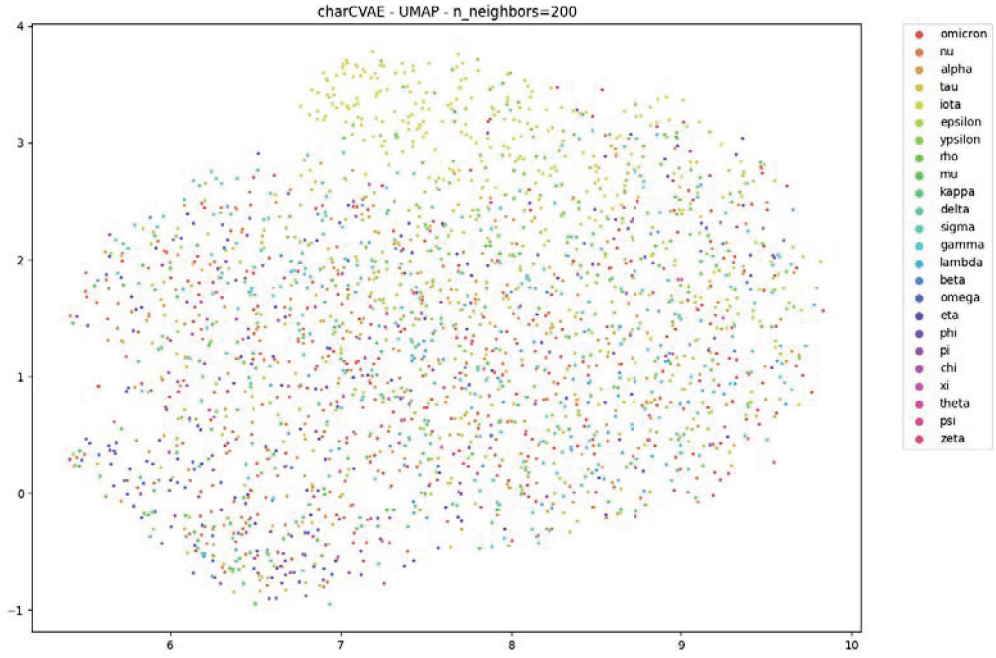


Figure A.35: UMAP Plot of the full 48-dimensional Space with 200 Neighbors

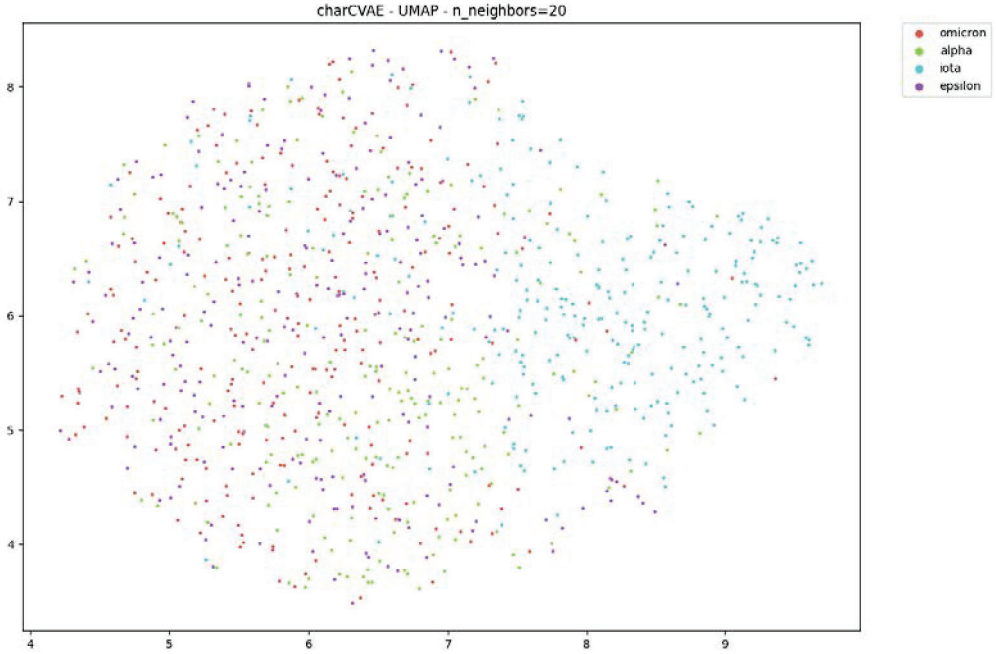


Figure A.36: UMAP Plot of the Latent Space of the most frequent Character Labels with 20 Neighbors

A.3.2 charCVAE - Clustering and Rand Index

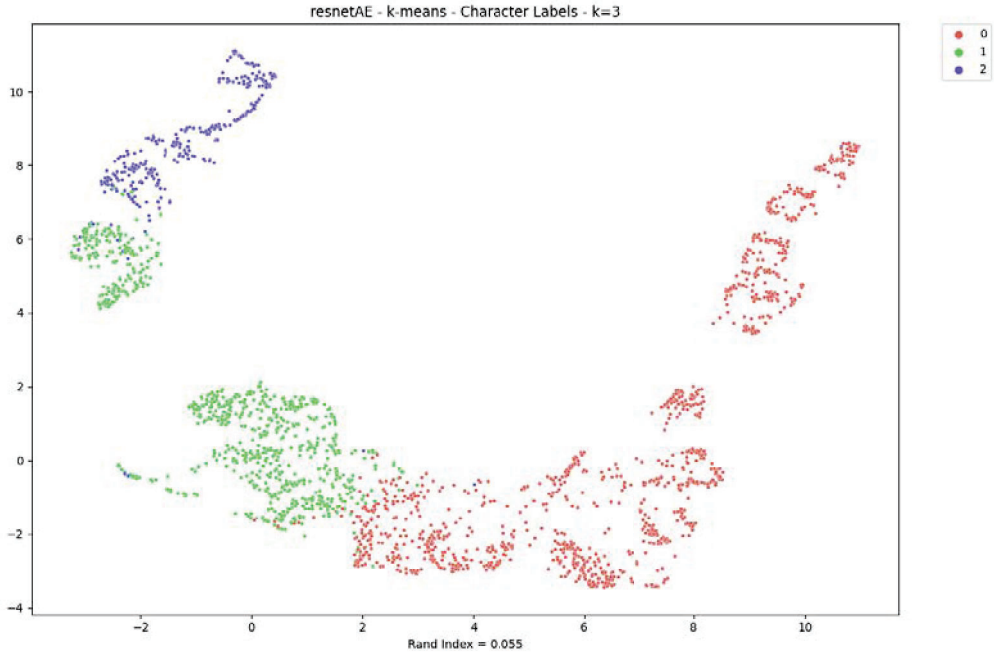


Figure A.37: K-Means with 3 Clusters on the full Latent Space

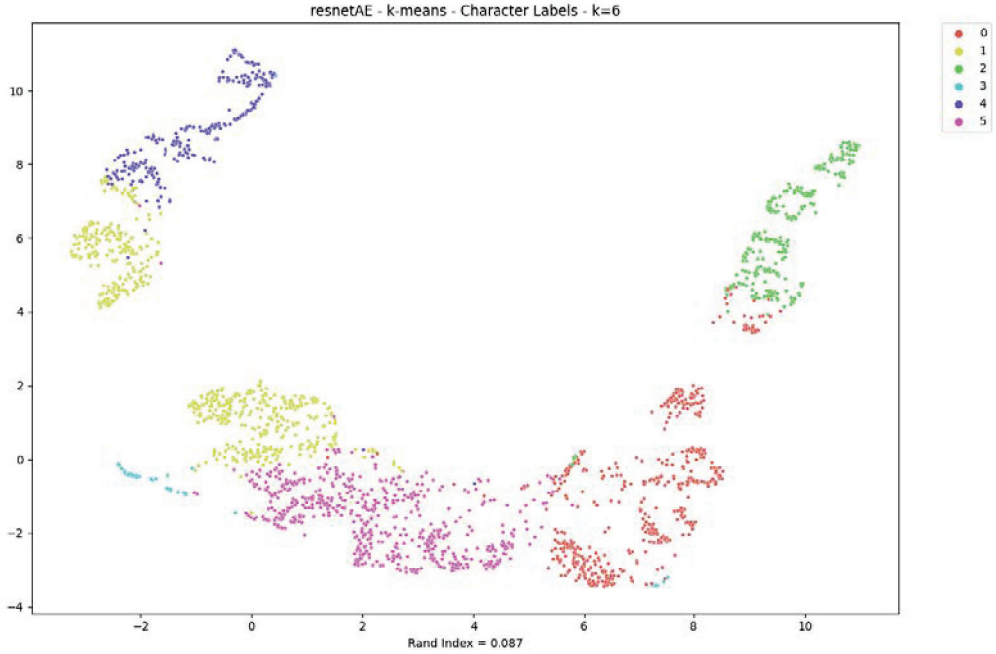


Figure A.38: K-Means with 6 Clusters on the full Latent Space

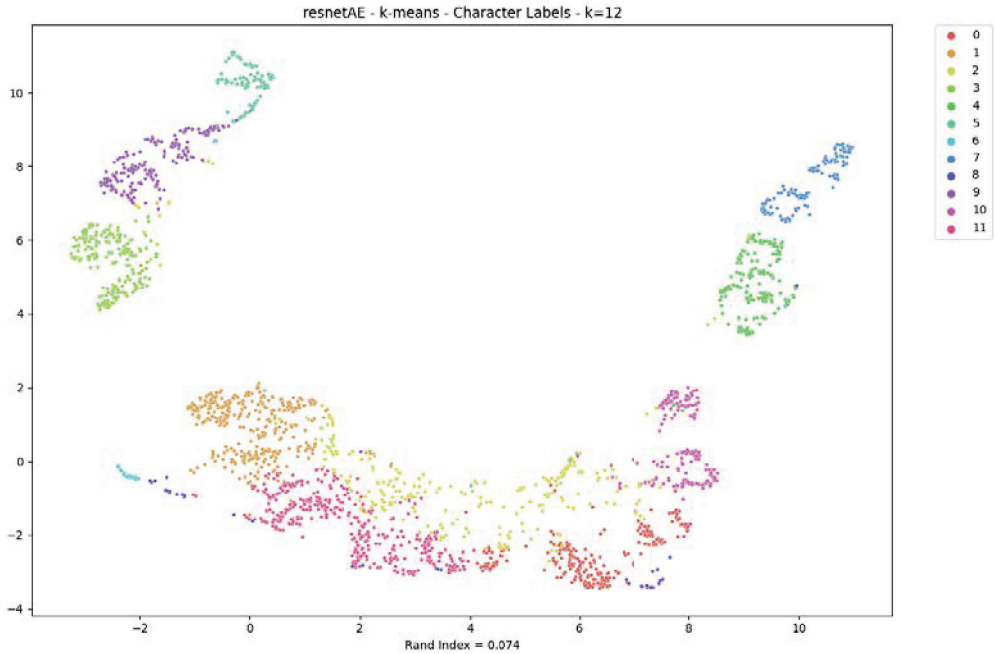
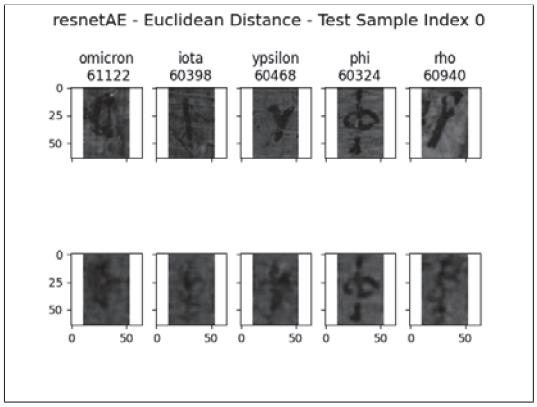
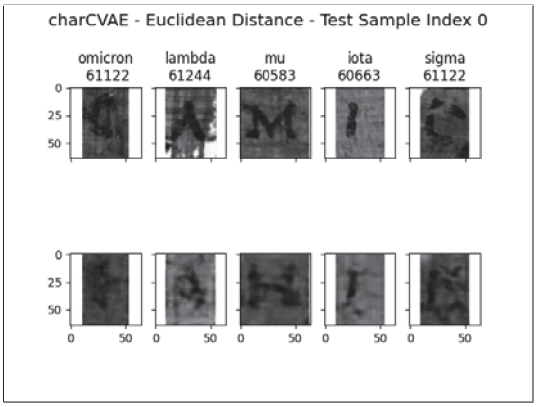


Figure A.39: K-Means with 12 Clusters on the full Latent Space

A.3.3 charCVAE - Euclidean Distance

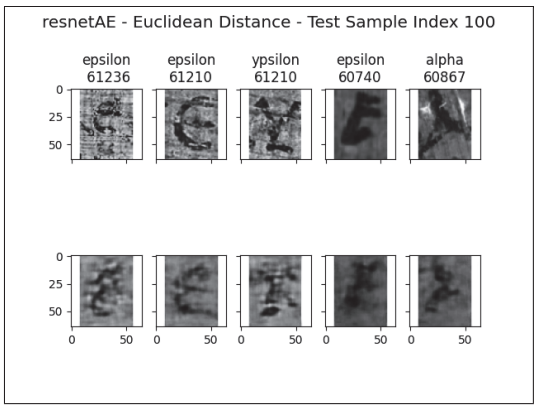


(a) resnetAE

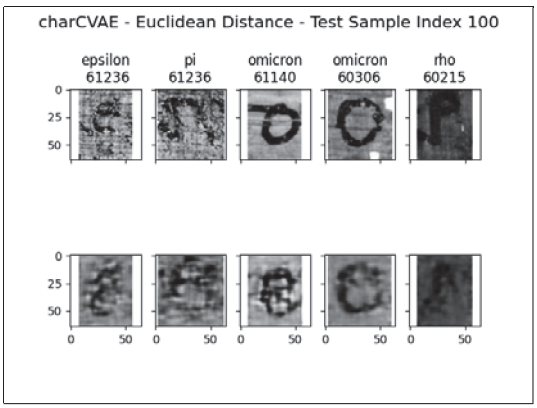


(b) charCVAE

Figure A.40: Sample 0 - resnetAE and charCVAE

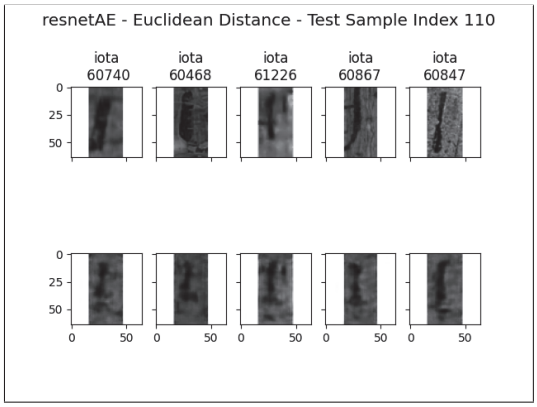


(a) resnetAE

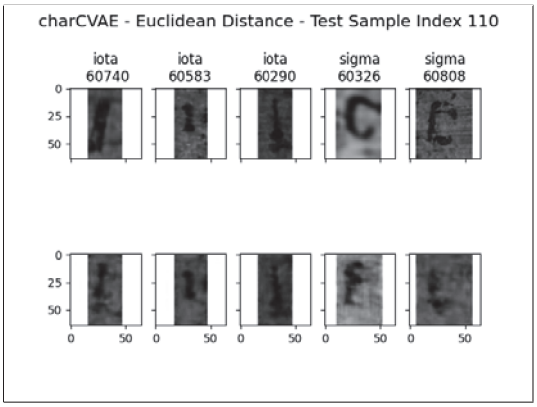


(b) charCVAE

Figure A.41: Sample 10 - resnetAE and charCVAE

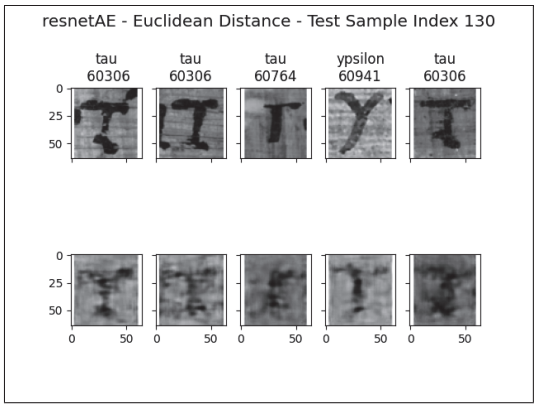


(a) resnetAE

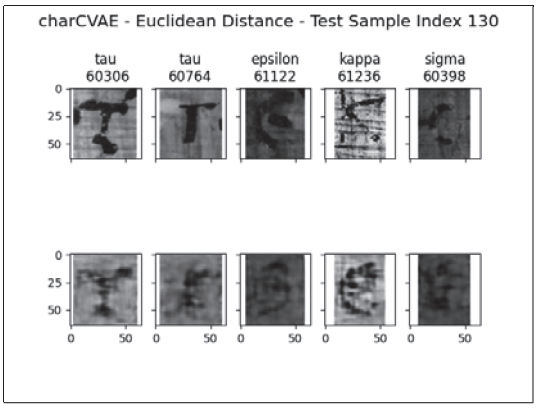


(b) charCVAE

Figure A.42: Sample 11 - resnetAE and charCVAE

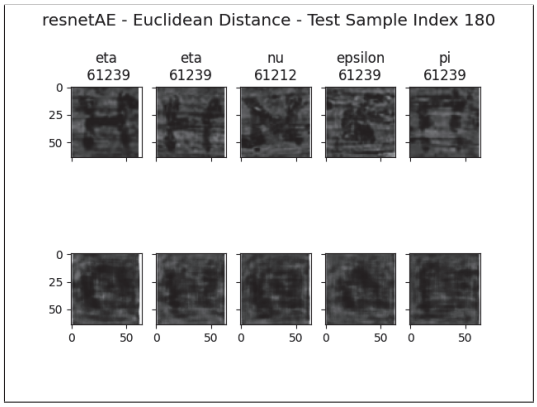


(a) resnetAE

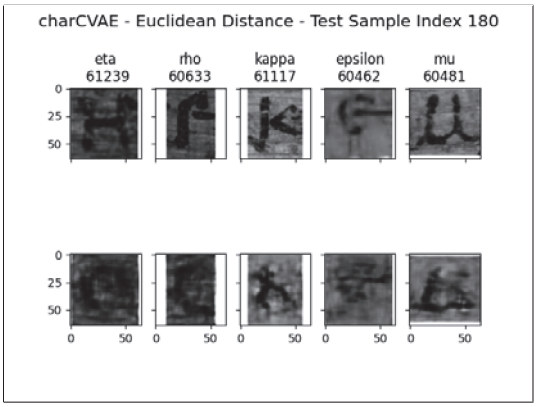


(b) charCVAE

Figure A.43: Sample 13 - resnetAE and charCVAE

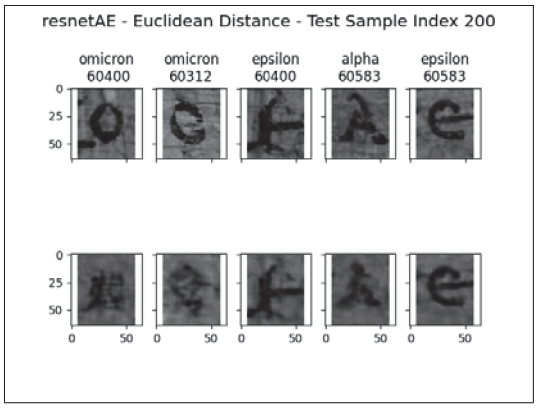


(a) resnetAE

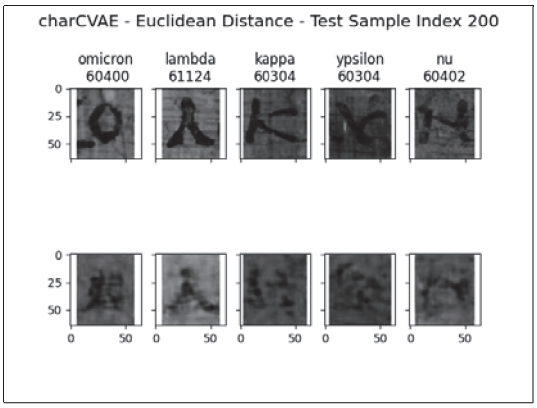


(b) charCVAE

Figure A.44: Sample 18 - resnetAE and charCVAE

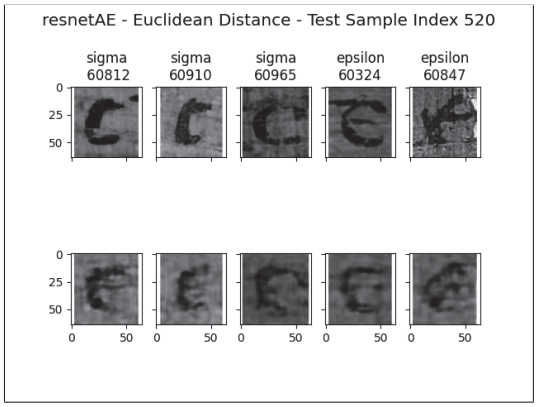


(a) resnetAE

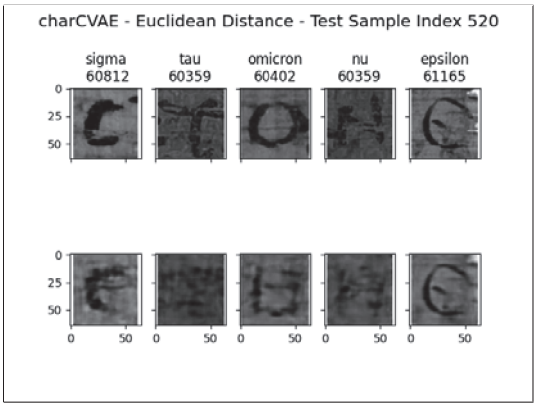


(b) charCVAE

Figure A.45: Sample 20 - resnetAE and charCVAE

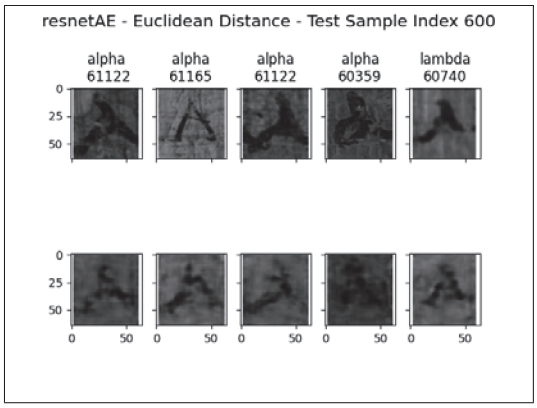


(a) resnetAE

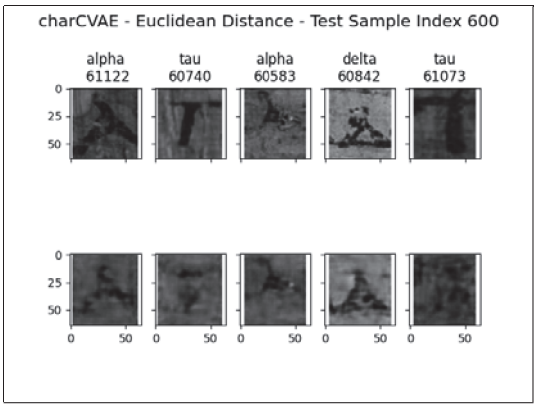


(b) charCVAE

Figure A.46: Sample 52 - resnetAE and charCVAE

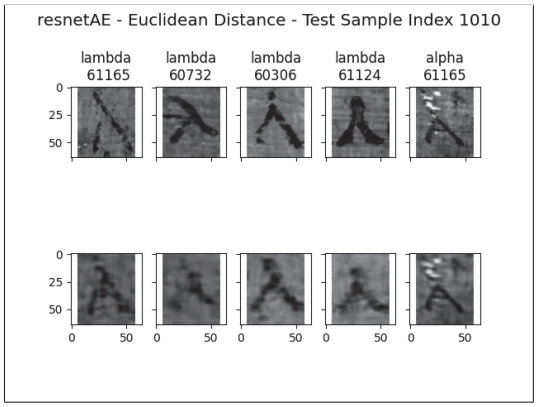


(a) resnetAE

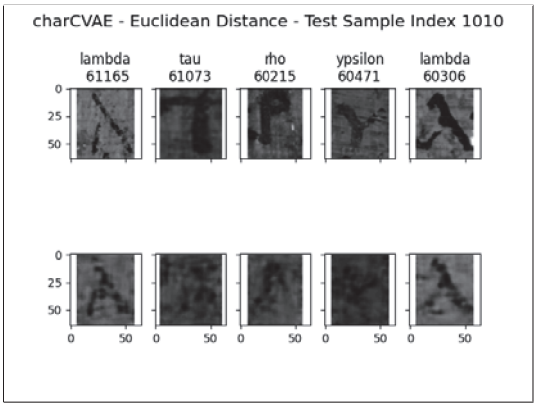


(b) charCVAE

Figure A.47: Sample 60 - resnetAE and charCVAE

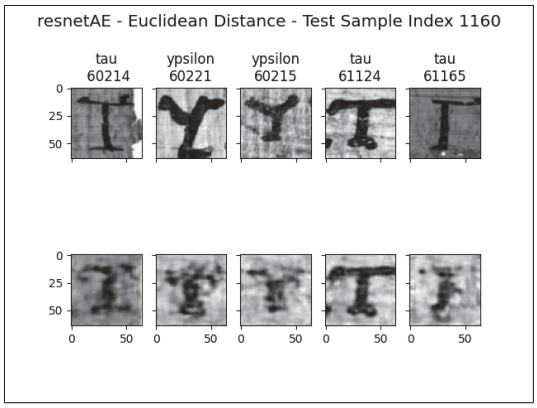


(a) resnetAE

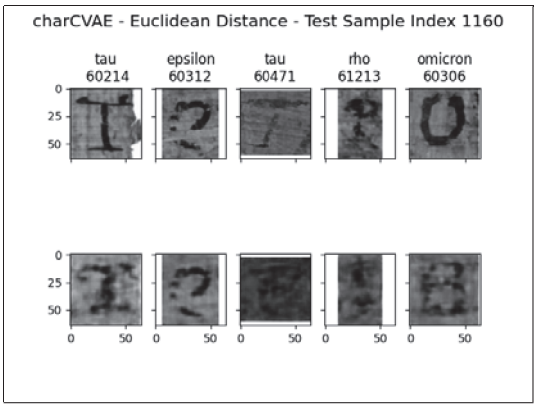


(b) charCVAE

Figure A.48: Sample 101 - resnetAE and charCVAE

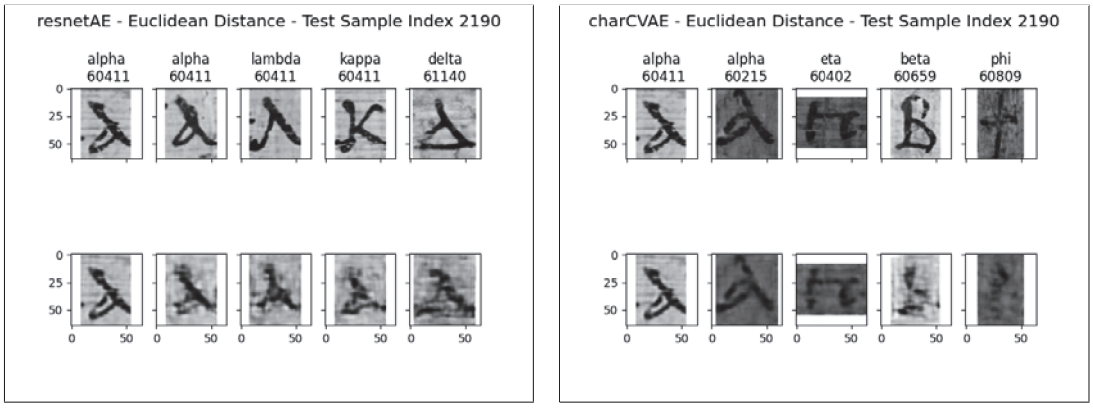


(a) resnetAE



(b) charCVAE

Figure A.49: Sample 116 - resnetAE and charCVAE



(a) resnetAE (b) charCVAE

Figure A.50: Sample 219 - resnetAE and charCVAE

A.4 Additional Material on fragCVAE Evaluation

A.4.1 fragCVAE - Dimensionality Reduction

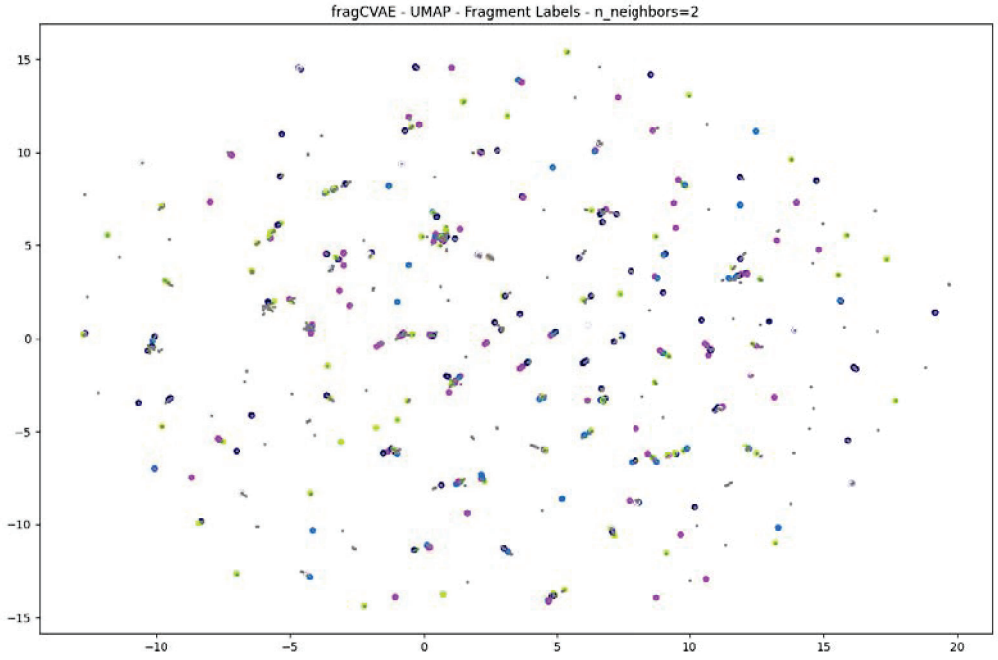


Figure A.51: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 2 Neighbors

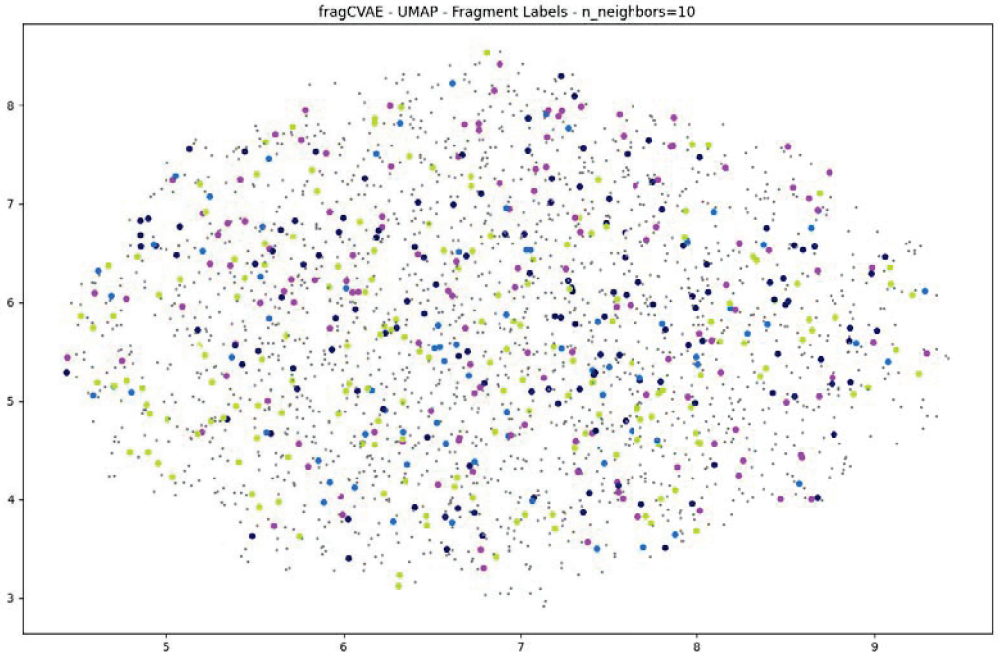


Figure A.52: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 10 Neighbors

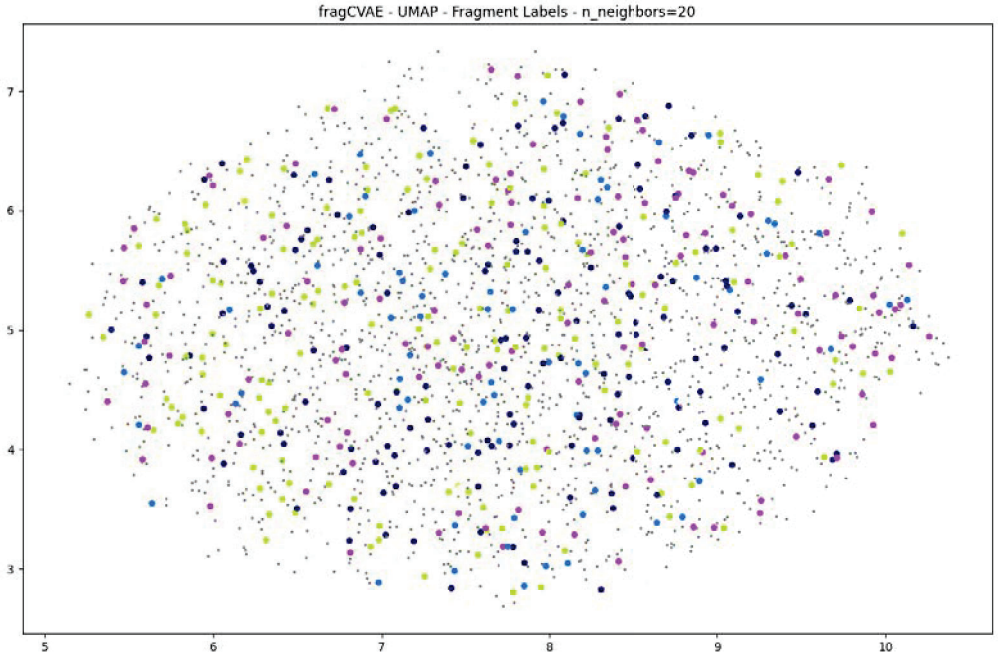


Figure A.53: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 20 Neighbors

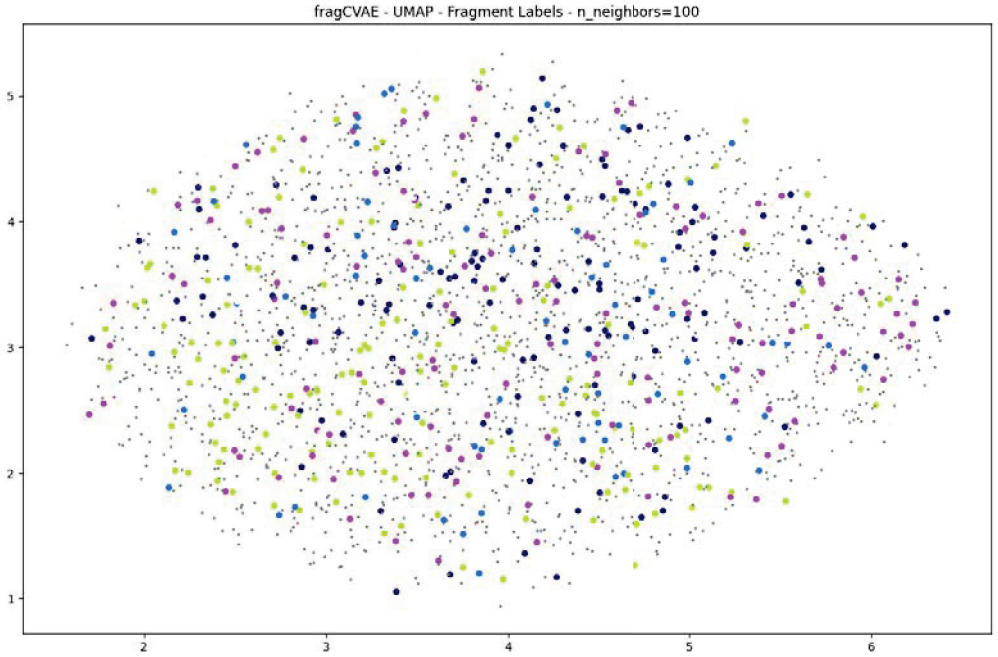


Figure A.54: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 100 Neighbors

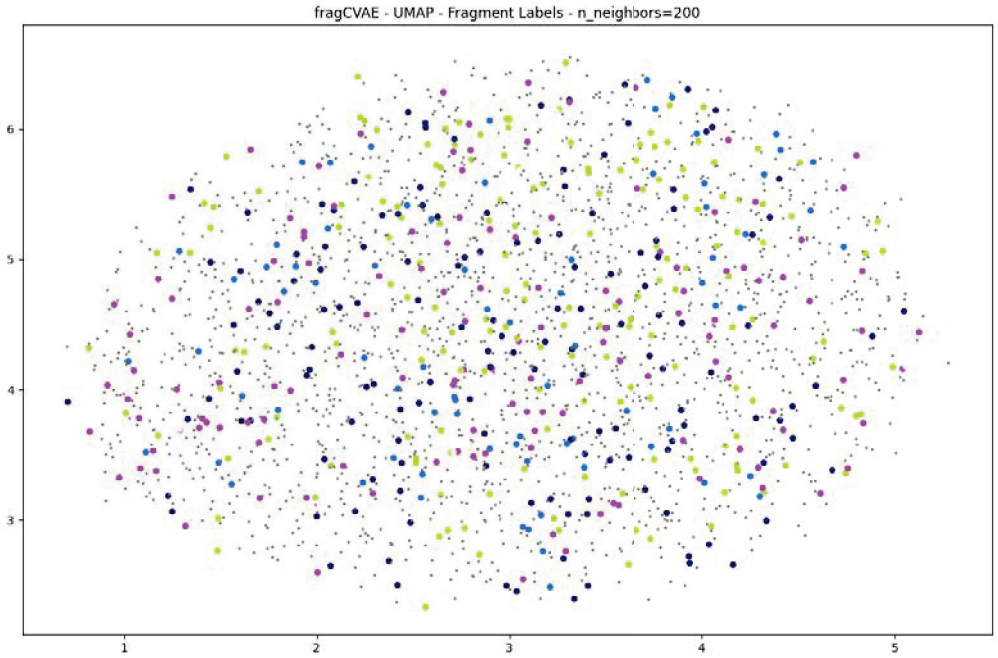
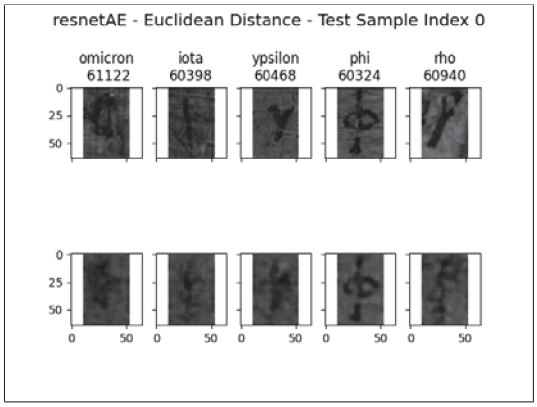


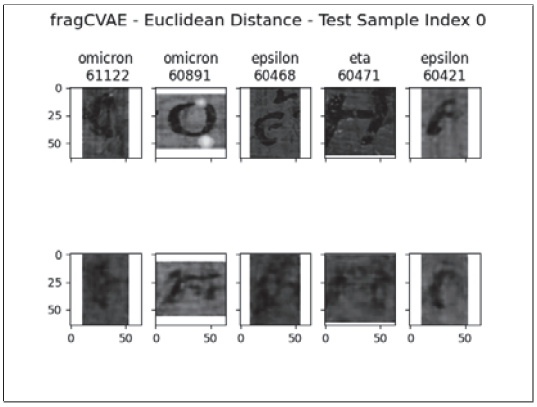
Figure A.55: UMAP Plot of the 48-dimensional Space of the six most frequent Fragment Labels with 200 Neighbors

A.4.2 fragCVAE - Clustering and Rand Index

A.4.3 fragCVAE - Dimensionality Reduction

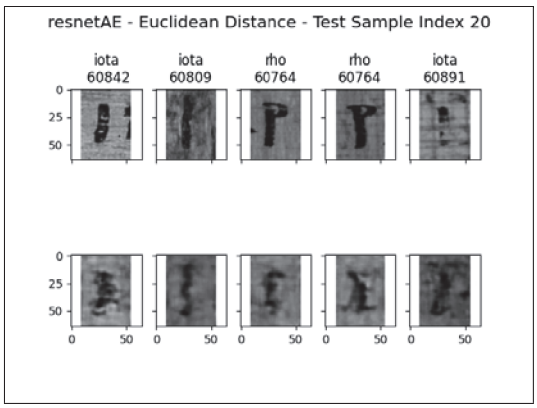


(a) resnetAE

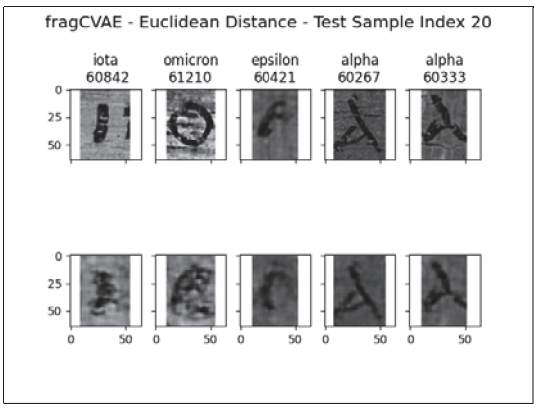


(b) fragCVAE

Figure A.56: Sample 0 - resnetAE and fragCVAE

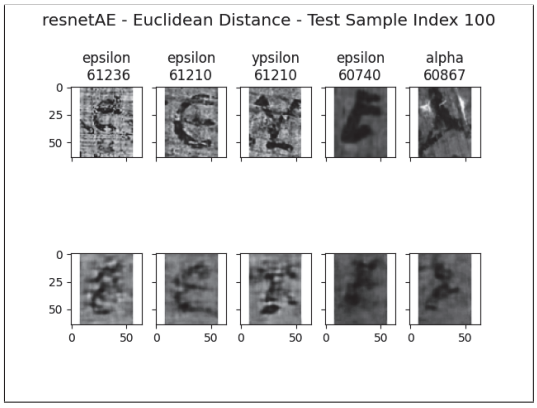


(a) resnetAE

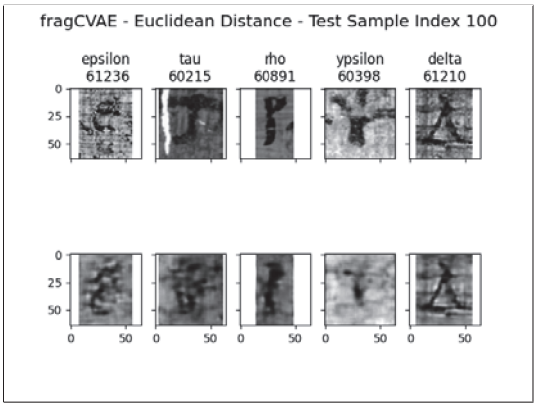


(b) fragCVAE

Figure A.57: Sample 2 - resnetAE and fragCVAE

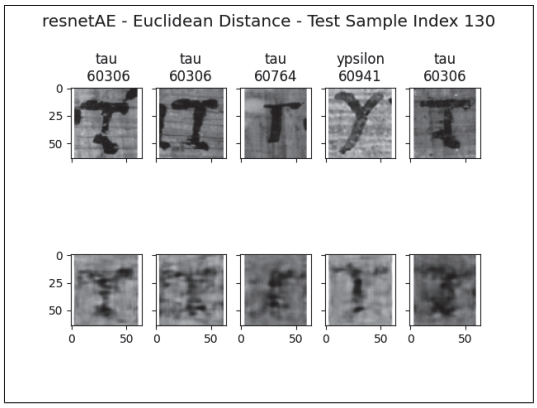


(a) resnetAE

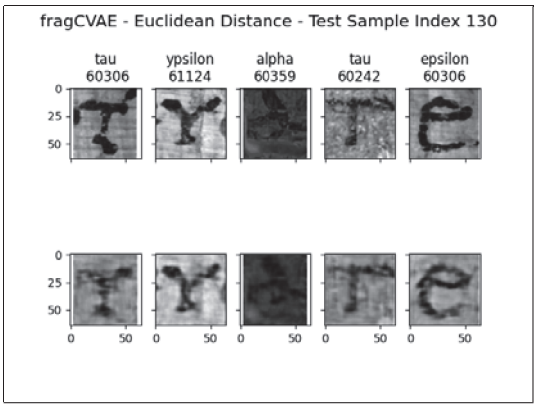


(b) fragCVAE

Figure A.58: Sample 10 - resnetAE and fragCVAE

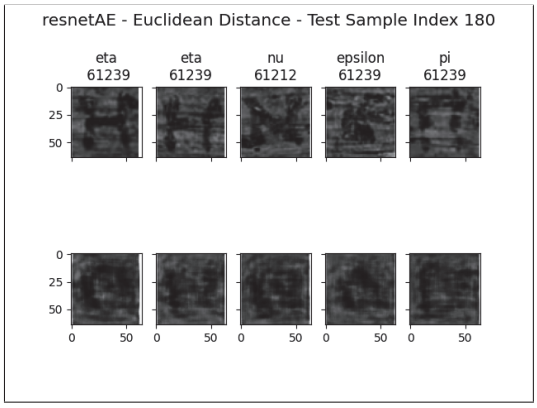


(a) resnetAE

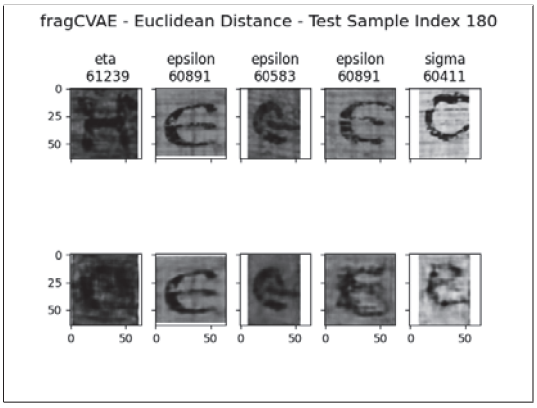


(b) fragCVAE

Figure A.59: Sample 13 - resnetAE and fragCVAE

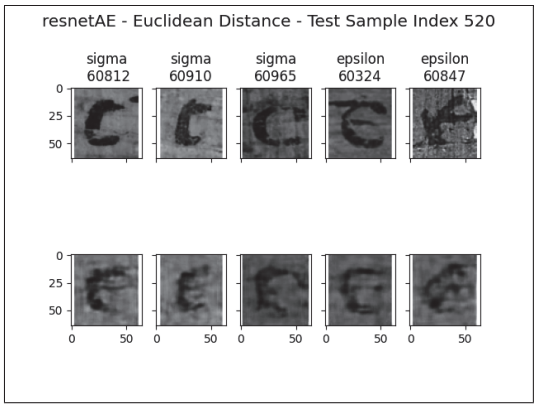


(a) resnetAE

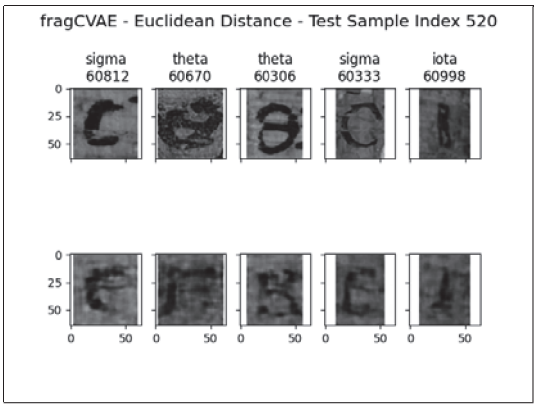


(b) fragCVAE

Figure A.60: Sample 18 - resnetAE and fragCVAE

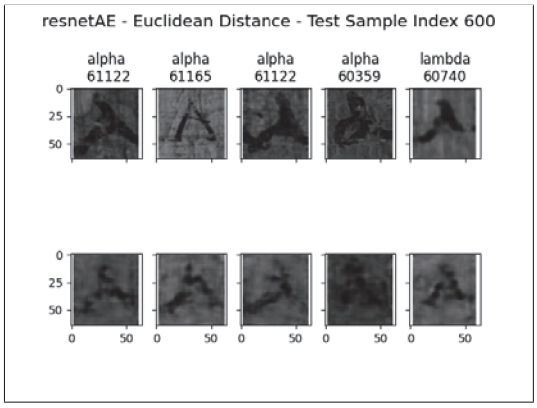


(a) resnetAE

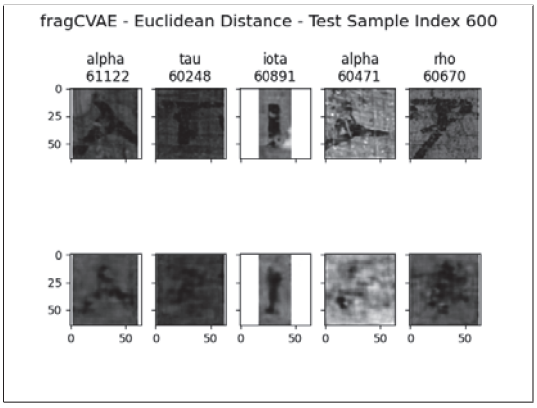


(b) fragCVAE

Figure A.61: Sample 52 - resnetAE and fragCVAE

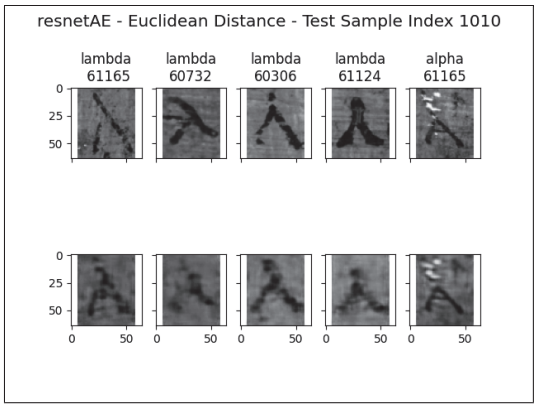


(a) resnetAE

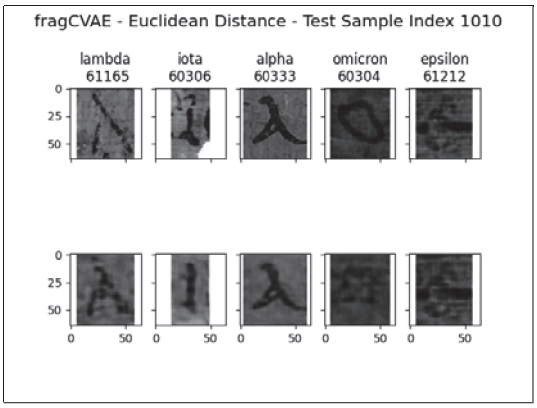


(b) fragCVAE

Figure A.62: Sample 60 - resnetAE and fragCVAE

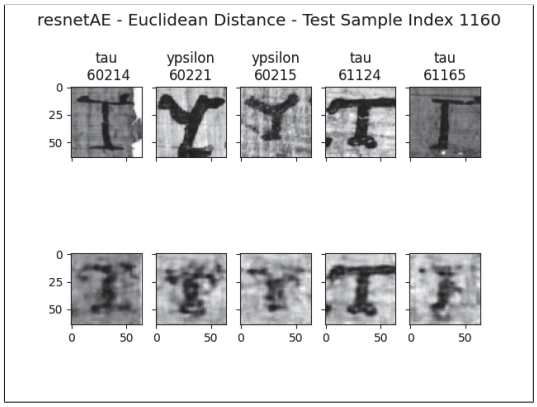


(a) resnetAE

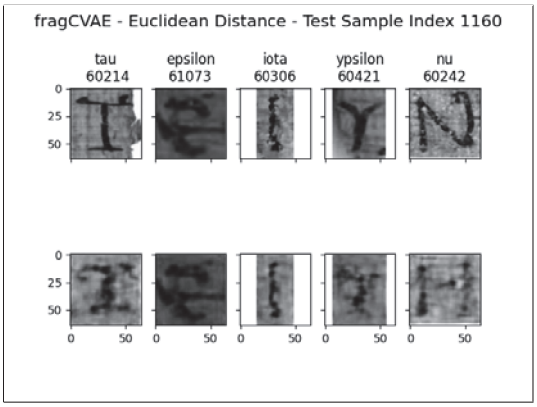


(b) fragCVAE

Figure A.63: Sample 101 - resnetAE and fragCVAE

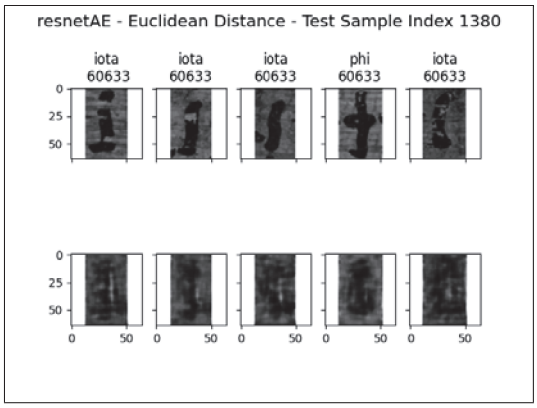


(a) resnetAE

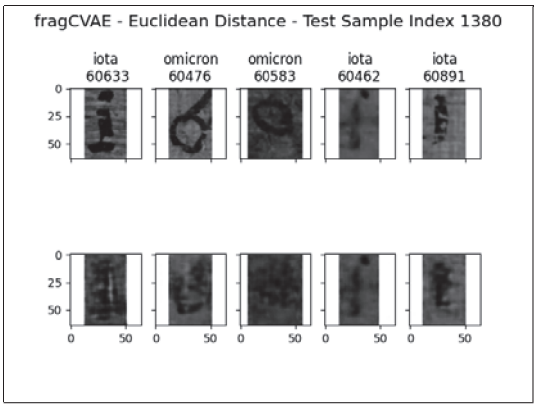


(b) fragCVAE

Figure A.64: Sample 116 - resnetAE and fragCVAE

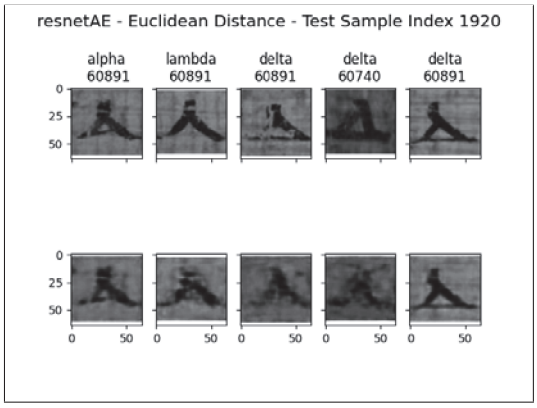


(a) resnetAE

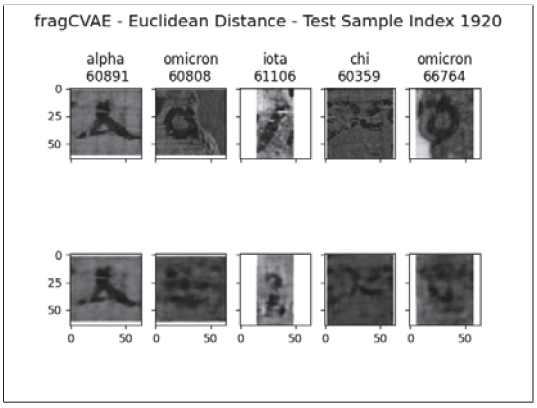


(b) fragCVAE

Figure A.65: Sample 138 - resnetAE and fragCVAE

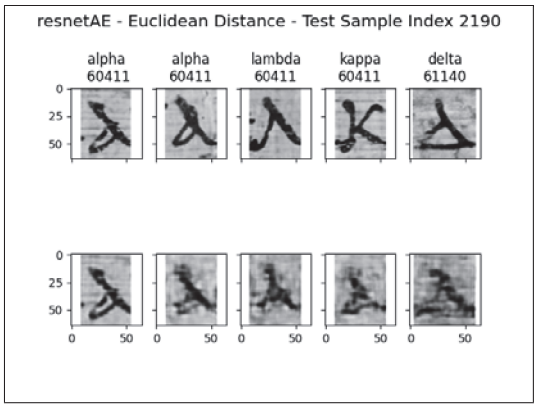


(a) resnetAE

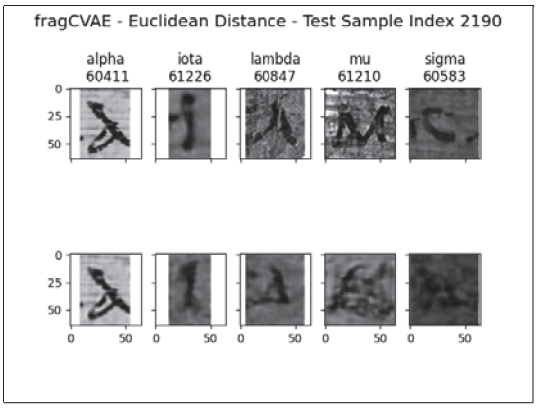


(b) fragCVAE

Figure A.66: Sample 192 - resnetAE and fragCVAE

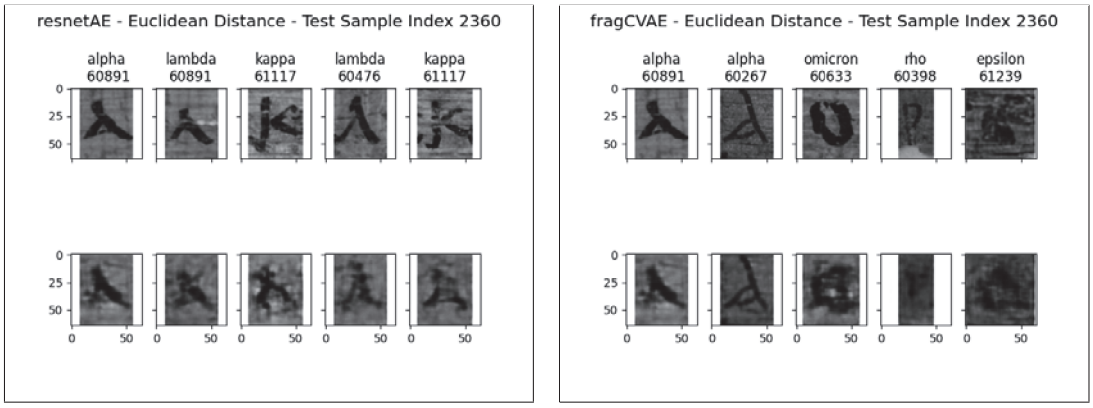


(a) resnetAE



(b) fragCVAE

Figure A.67: Sample 219 - resnetAE and fragCVAE



(a) resnetAE (b) fragCVAE

Figure A.68: Sample 236 - resnetAE and fragCVAE

University of Basel

Faculty of Science

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)
Translation from German original

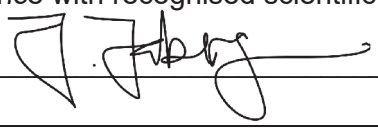
Title of Thesis:

Name Assesor: Ivan Dokmanic

Name Student: Jannik Jaberg

Matriculation No.: 17-054-370

With my signature I declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Place, Date: 14.9.2022 Student: 

Will this work be published?

- [ ] No
- [ ] Yes. With my signature I confirm that I agree to a publication of the work (print/digital) ~~in the library, on the research database of the University of Basel and/or~~ on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of:

Place, Date: Basel, 14.9.2022 Student: 

Place, Date: 14.9.2022 Assessor: 

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .